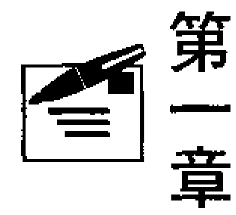
本书常用符号说明

$\boldsymbol{W}$	权值矩阵
$\boldsymbol{W}(k)$	第 k 个时刻或第 k 次迭代时的权值矩阵
$\boldsymbol{w}_i$	权值矩阵 $\boldsymbol{W}$ 中的第 i 行矢量
$w_{i,j}$	权值矩阵 $\boldsymbol{W}$ 中的第 i 行第 j 列元素
$\boldsymbol{p}$	输入矢量
$\boldsymbol{p}(k)$	第 k 个时刻的输入矢量
p_i	输入矢量 $\boldsymbol{p}$ 中的第 i 个元素
$\boldsymbol{b}$	阈值矢量
$\boldsymbol{b}(k)$	第 k 个时刻或第 k 次迭代时的阈值矢量
b_i	阈值矢量 $\boldsymbol{b}$ 中的第 i 个元素
f	神经元传递函数
$\boldsymbol{IW}^{k,l}$	网络的第 l 个输入到网络第 k 层神经元的连接权值矩阵
$\boldsymbol{LW}^{k,l}$	网络的第 l 层神经元到第 k 层神经元的连接权值矩阵
$\boldsymbol{W}^i$	单输入级联网络的连接权值矩阵：当 $i=1$ 时，$\boldsymbol{W}^i$ 是输入权值矩阵；当 $i\neq 1$ 时，$\boldsymbol{W}^i$ 是网络第 $i-1$ 层神经元到第 i 层神经元的连接权值矩阵
$\boldsymbol{W}^{\mathrm{c}}$	反馈连接权值矩阵
S^i	网络第 i 层神经元的个数

目　录



第一章 绪 论

1.1 神经网络的发展概况

研究人类智能一直是科学发展中最有意义、最激动人心，也是最富有挑战性的课题。人工神经网络（Artificial Neural Network－ANN），简称为“神经网络”（NN），作为对人脑最简单的一种抽象和模拟，是探索人类智能奥秘的有力工具。神经网络技术作为智能科学的领头羊，是近年来发展起来的一门十分活跃的交叉学科。它涉及生物、电子、计算机、数学、物理等学科，有着广泛的应用前景。

神经网络的研究始于 20 世纪 40 年代。半个多世纪以来，它经历了一条由兴起到衰退、又由衰退到兴盛的曲折发展过程，这一发展过程大致可以分为以下四个阶段。

1. 初始发展阶段

1943 年，心理学家 W. S. McCulloch 和数学家 W. Pitts 在研究生物神经元的基础上提出了一种简单的人工神经元模型，即后来所谓的“M-P 模型”，在该模型中，神经元的活动表现为“兴奋”和“抑制”两个状态，其基本工作原理与现在的阈值单元模型基本相同。虽然 M-P 模型过于简单，且只能完成一些简单的逻辑运算，但它的出现开创了神经网络研究的先河，并为以后的研究提供了依据。

1949 年，心理学家 D. O. Hebb 发表了论著《行为自组织》，首先提出了一种调整神经网络连接权值的规则。他认为，学习过程是在突触上发生的，连接权值的调整正比于两相连神经元活动状态的乘积，这就是著名的 Hebb 学习律。直到现在，Hebb 学习律仍然是神经网络中一个极为重要的学习规则。

1957 年，F. Rosenblatt 提出了著名的感知器（Perceptron）模型，这是第一个真正的人工神经网络。这个模型由简单的阈值神经元构成，初步具备了诸如并行处理、分布存储和学习等神经网络的一些基本特性，从而确立了从系统角度研究神经网络的基础。

1960 年，B. Widrow 和 M. E. Hoff 提出了自适应线性单元（Adaline）网络，不仅在计算机上对该网络进行了模拟，而且还做成了硬件。同时，他们还提出了 Widrow-Hoff 学习算法，改进了网络权值的学习速度和精度，后来这个算法被称为 LMS 算法，即数学上的最速下降法，这种算法在以后的 BP 网络及其他信号处理系统中得到了广泛的应用。

2. 低潮时期

1969 年，美国麻省理工学院著名的人工智能专家 M. Minsky 和 S. Papert 共同出版了名



为《感知器》的专著，指出单层的感知器网络只能用于线性问题的求解，而对于像 XOR（异或）这样简单的非线性问题却无法求解。他们还指出，能够求解非线性问题的网络，应该是具有隐层的多层神经网络，而将感知器模型扩展到多层网络是否有意义，还不能从理论上得到有力的证明。Minsky 的悲观结论对当时神经网络的研究是一个沉重的打击。由于当时计算机技术还不够发达，VLSI 尚未出现，神经网络的应用还没有展开，而人工智能和专家系统正处于发展的高潮，因而很多人放弃了对神经网络的研究，致使在这以后的 10 年中，神经网络的研究进入了一个缓慢发展的低潮期。

虽然在整个 20 世纪 70 年代，对神经网络理论的研究进展缓慢，但并没有完全停顿下来。世界上一些对神经网络抱有坚定信心和严肃科学态度的学者一直没有放弃他们的努力。许多神经网络模型，如线性神经网络模型、自组织识别神经网络模型以及将神经元的输出函数与统计力学中的玻耳兹曼分布相联系的 Boltzmann 机模型等，都是在这个时期出现的。

3. 复兴时期

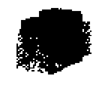
美国加州理工学院生物物理学家 John. J. Hopfield 博士在 1982 年和 1984 年先后发表了两篇十分重要的论文，在他所提出的 Hopfield 网络模型中首次引入了网络能量的概念，并给出了网络稳定性判据。Hopfield 网络不仅在理论分析与综合上均达到了相当的深度，最有意义的是该网络很容易用集成电路来实现。Hopfield 网络引起了许多科学家的理解与重视，也引起了半导体工业界的重视。1984 年，AT&T Bell 实验室宣布利用 Hopfield 理论研制成功了第一个硬件神经网络芯片。尽管早期的 Hopfield 网络还存在一些问题，但不可否认，正是由于 Hopfield 的研究才点亮了神经网络复兴的火把，从而掀起了神经网络研究的热潮。

如果说 Hopfield 的研究成果打破了神经网络理论 10 年徘徊的局面，那么 1986 年 D. E. Rumelhart 和 J. L .McCelland 及其研究小组提出的 PDP（Parallel Distributed Processing）网络思想，则为神经网络研究新高潮的到来起到了推波助澜的作用。尤其是他们提出的误差反向传播算法，即 BP 算法，已成为至今影响最大、应用最广的一种网络学习算法。

4. 发展高潮期

20 世纪 80 年代中期以来，神经网络的应用研究取得了很大的成绩，涉及面非常广泛。为了适应人工神经网络的发展，1987 年成立了国际神经网络学会，并于同年在美国圣地亚哥召开了第一届国际神经网络会议。此后，神经网络技术的研究始终呈现出蓬勃活跃的局面，理论研究不断深入，应用范围不断扩大。尤其是进入 20 世纪 90 年代，随着 IEEE 神经网络会刊的问世，各种论文专著逐年增加，在全世界范围内逐步形成了研究神经网络前所未有的新高潮。

从众多神经网络的研究和应用成果不难看出，神经网络的发展具有强大的生命力。尽管当前神经网络的智能水平还不高，许多理论和应用性问题还未得到很好的解决，但是，随着人们对大脑信息处理机制认识的日益深化，以及不同智能学科领域之间的交叉与渗透，人工神经网络必将对智能科学的发展发挥更大的作用。



1.2　神经网络的应用及研究方向

神经网络作为一种新的方法体系，具有分布并行处理、非线性映射、自适应学习和鲁棒容错等特性，这使得它在模式识别、控制优化、智能信息处理以及故障诊断等方面都有广泛的应用。

目前，神经网络的研究可以分为理论研究和应用研究两大方面。

理论研究可分为以下两类：

(1) 利用神经生理与认知科学研究人脑思维及智能机理。

(2) 利用神经科学基础理论的研究成果，用数理方法探索智能水平更高的人工神经网络模型，深入研究网络的算法和性能（如稳定性、收敛性、容错性、鲁棒性等），开发新的网络数理理论（如神经网络动力学、非线性神经场等）。

应用研究可分为以下两类：

(1) 神经网络的软件模拟和硬件实现的研究。

(2) 神经网络在各个领域中应用的研究，这些领域主要包括模式识别、信号处理、知识工程、专家系统、优化组合、智能控制等。

随着神经网络理论本身以及相关理论、相关技术的不断发展，神经网络的应用必将更加深入和广泛。

1.3　MATLAB 6.x 神经网络工具箱（4.0 版本）简介

MATLAB 神经网络工具箱 4.0 版本是 Mathworks 公司最新推出的 MATLAB 6.x 高性能可视化数值计算软件的组成部分。它主要针对神经网络系统的分析与设计，提供了大量可供直接调用的工具箱函数、图形用户界面和 Simulink 仿真工具，是进行神经网络系统分析与设计的绝佳工具。与以往的 MATLAB 神经网络工具箱相比，神经网络工具箱 4.0 版本出现了许多新增功能，使用起来更加方便。这些新增功能主要包括：

- 新增了三个专门用于基于神经网络控制系统应用与设计的 Simulink 模块：神经网络预测控制器模块、反馈线性化控制器模块和模型参考自适应控制器模块，极大地方便了基于神经网络的控制系统设计与仿真。
- 提供了神经网络设计与仿真 GUI，使用户能够方便地通过图形用户界面进行神经网络的设计与仿真。
- 改进并新增了四个神经网络训练函数：trainb、trainc、trainr、trains，其中 trainb 用于批量式训练，其他三个函数则用于渐进式训练。
- 改进了线性神经网络设计函数 newlind，使其能够对多输入、多输出的线性神经网络进行直接设计。
- 改进的“提前停止”算法可以和贝叶斯正则化方法联合使用，从而可以更好地提高神经网络的推广或泛化能力。





第二章 基于 MATLAB 的神经网络基本理论

2.1 神经网络的基本概念

2.1.1 神经元模型

神经网络的基本单元称为神经元，它是对生物神经元的简化与模拟。神经元的特性在某种程度上决定了神经网络的总体特性。大量简单神经元的相互连结即构成了神经网络。一个典型的具有 R 维输入的神经元模型可以用图 2.1 来加以描述。

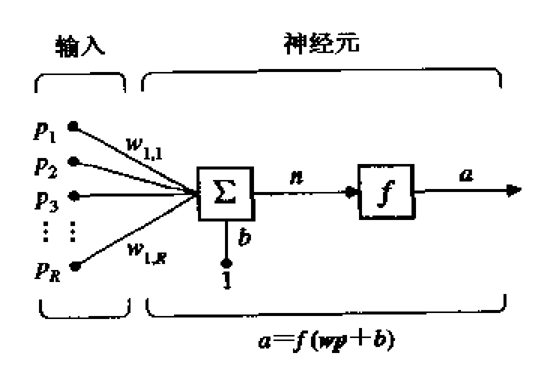


图 2.1 神经元模型

由图 2.1 可见，一个典型的神经元模型主要由以下五部分组成。

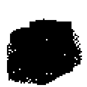
(1) 输入。

$p_1,p_2,\cdots,p_R$ 代表神经元的 R 个输入。在 MATLAB 中，输入可以用一个 $R\times1$ 维的列矢量 $\boldsymbol{p}$ 来表示（其中 T 表示取转置）

$$\boldsymbol{p}=[p_1,p_2,\cdots,p_R]^{\mathrm{T}}$$

(2) 网络权值和阈值。

$w_{1,1},w_{1,2},\cdots,w_{1,R}$ 代表网络权值，表示输入与神经元间的连接强度；b 为神经元阈值，可以看作是一个输入恒为 1 的网络权值。在 MATLAB 中，神经元的网络权值可以用一个 $1\times R$ 维的行矢量 $\boldsymbol{w}$ 来表示：



$$w=[w_{1,1},w_{1,2},\cdots,w_{1,R}]$$

阈值 b 为 1×1 的标量。

值得注意的是，不论是网络权值还是阈值都是可调的。正是基于神经网络权值和阈值的动态调节，神经元乃至神经网络才得以表现出某种行为特性。因此，网络权值和阈值的可调性是神经网络学习特性的基本内涵之一。

(3) 求和单元。

求和单元完成对输入信号的加权求和，即

$$n=\sum_{i=1}^{R}p_i w_{1,i}+b$$

这是神经元对输入信号处理的第一个过程。在 MATLAB 语言中，该过程可以通过输入矢量和权值矢量的点积形式加以描述，即

```
n = w * p + b
```

(4) 传递函数。

在图 2.1 中，f 表示神经元的传递函数或激发函数，它用于对求和单元的计算结果进行函数运算，得到神经元的输出，这是神经元对输入信号处理的第二个过程。表 2-1 给出了几种典型的神经元传递函数形式及描述。

表 2-1　几种典型的神经元传递函数形式

传递函数名称	函数表达式	函数曲线	MATLAB 函数
阈值函数	$f(x)=\begin{cases}1 & x\geq0\\0 & x<0\end{cases}$	a, +1, 0, n, −1 a=hardlim(n)	hardlim
线性函数	$f(x)=kx$	a, +1, 0, n, −1 a=purelin(n)	purelin
对数Sigmoid函数	$f(x)=1/(1+e^{-x})$	a, +1, 0, n, −1 a=logsig(n)	logsig
正切Sigmoid函数	$f(x)=\tanh(x)$	a, +1, 0, n, −1 a=tansig(n)	tansig



(5) 输出。

输入信号经神经元加权求和及传递函数作用后，得到最终的输出为

$$a = f(wp + b)$$

若取传递函数为 hardlim 函数，则神经元输出可用 MATLAB 语句表示为

```
a = hardlim ( w * p + b )
```

为了描述方便，图 2.1 所示神经元模型可以用图 2.2 所示的缩略形式加以描述。

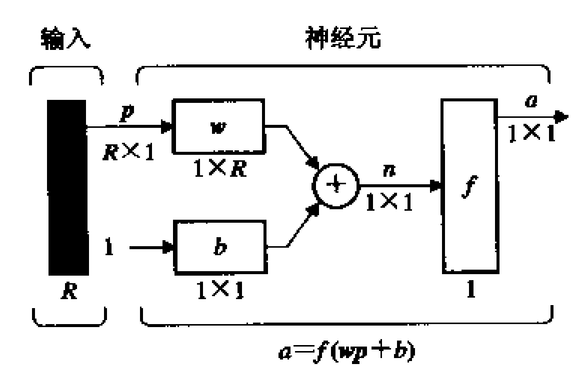


图 2.2　神经元模型的缩略形式

在图 2.2 中，黑色矩形块代表神经元的输入矢量，R 为输入矢量的维数；⊕代表加权求和运算单元；f 为传递函数运算单元。可见，该图简洁清晰地描述了神经元的结构特性及其对输入信号的处理过程。以后各章节中均采用这种形式对神经元或神经网络进行描述。

2.1.2　神经网络的结构与类型

1. 神经网络的基本结构及描述

神经网络是由大量简单神经元相互连接构成的复杂网络。一个典型的具有 R 维输入、S 个神经元的单层神经网络模型可以用图 2.3 来加以描述。

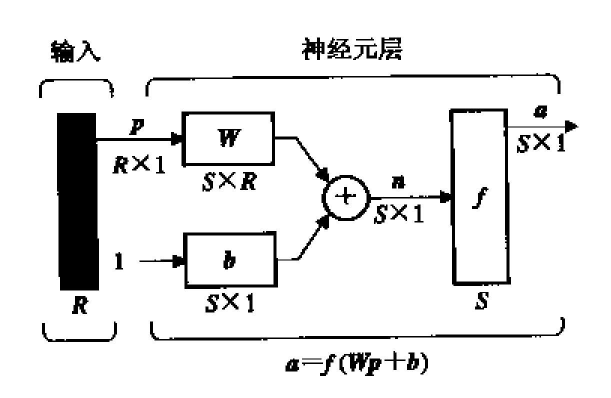
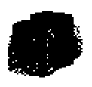


图 2.3　单层神经网络模型

在图 2.3 中，$\boldsymbol{p}$ 为 $R\times1$ 维的输入矢量，网络层由权值矩阵 $\boldsymbol{W}(S\times R)$、阈值矢量 $\boldsymbol{b}(S\times1)$、求和单元⊕和传递函数运算单元 f 组成，S 个神经元的输出组成了 $S\times1$ 维的神经网络输出矢量 $\boldsymbol{a}$



$$a = f(Wp + b)$$

其中，输入层网络权值矩阵 **W** 和阈值矢量 **b** 的具体形式如下

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \cdots & \cdots & \cdots & \cdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_S \end{bmatrix}$$

在单层神经网络基础上可以构造多层神经网络。一个典型的三层神经网络模型如图 2.4 所示。

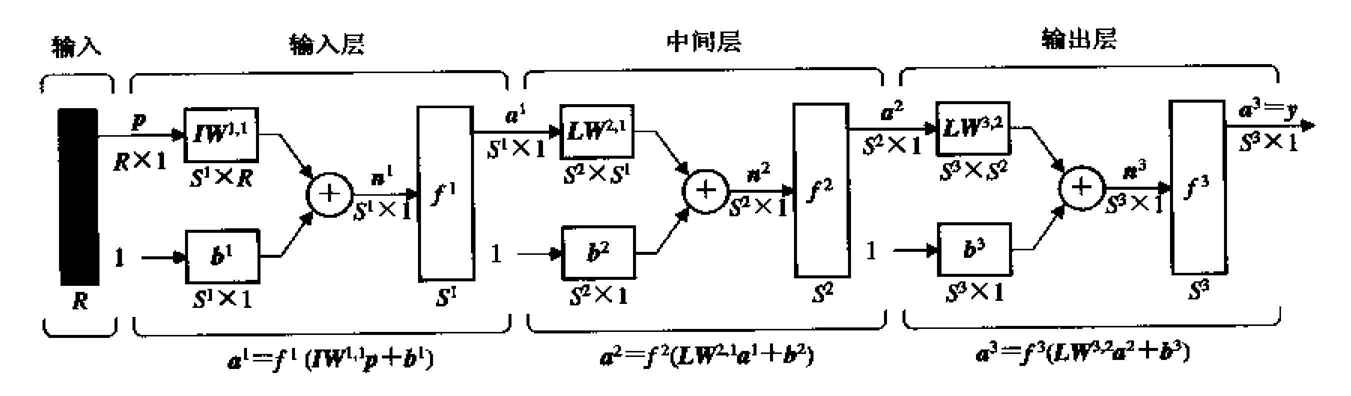


图 2.4　三层神经网络模型

产生神经网络最终输出的网络层称为输出层，输入层和中间层也称为隐层。图 2.4 所示的神经网络三层神经元的数目分别为：S^1、S^2、S^3。为了描述方便，采用上标法对神经网络中相关元素加以标记，其中，$IW^{1,1}(S^1 \times R)$表示输入层权值矩阵，$LW^{2,1}(S^2 \times S^1)$和 $LW^{3,2}(S^3 \times S^2)$分别表示第一层到第二层、第二层到第三层的网络层权值矩阵。b^1、b^2、b^3 分别表示各层的网络阈值矢量。神经网络的输出为

$$a^3 = f^3(LW^{3,2} f^2(LW^{2,1} f^1(IW^{1,1}p + b^1) + b^2) + b^3) = y$$

2. 神经网络的分类

神经网络的类型多种多样，它们是从不同角度对生物神经系统不同层次的抽象和模拟。从功能特性和学习特性来分，典型的神经网络模型主要包括感知器、线性神经网络、BP 网络、径向基函数网络、自组织映射网络和反馈神经网络等。一般来说，当神经元的模型确定之后，一个神经网络的特性及其功能主要取决于网络的拓扑结构及学习方法。从网络拓扑结构角度来看，神经网络可以分为以下四种基本形式。

(1) 前向网络。

前向网络结构如图 2.5（*a*）所示。网络中的神经元是分层排列的，每个神经元只与前一层神经元相连。最上一层为输出层，最下一层为输入层。输入层和中间层也称为隐层。隐层的层数可以是一层或多层，前向网络在神经网络中应用十分广泛，感知器、线性网络、BP 网络都属于这种类型。



(2) 从输出到输入有反馈的前向网络。

从输出到输入有反馈的前向网络结构如图 2.5（*b*）所示。该网络本身是前向型的，但是与上一种不同的是从输出到输入有反馈回路。Fukushima 网络就属于这种类型。

(3) 层内互连前向网络。

层内互连前向网络结构如图 2.5（*c*）所示。通过层内神经元的相互连接，可以实现同一层神经元之间的相互制约，从而可以将层内神经元分为几组，让每组作为一个整体来动作。一些自组织竞争型神经网络就属于这种类型。

(4) 互连网络。

互连网络结构如图 2.5（*d*）所示。互连网络又分为局部互连和全互连两种。全互连网络中每个神经元的输出都与其他神经元相连，而局部互连网络中，有些神经元之间没有连接关系。Hopfield 网络和 Boltzmann 网络就属于这一网络类型。

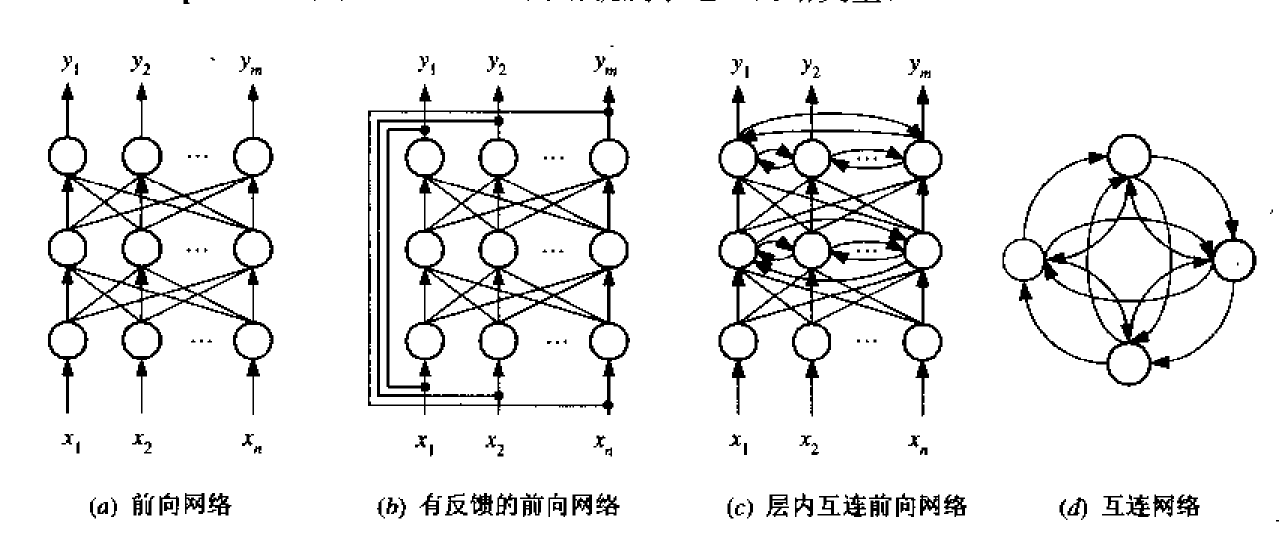


图 2.5　神经网络的典型拓扑结构

此外，根据网络输入或网络层中是否含有延迟或反馈环节，还可把神经网络分为静态网络和动态网络。动态网络中通常含有网络输入延迟或反馈环节，比如自适应滤波线性神经网络和 Hopfield 网络；静态神经网络中则不含任何延迟或反馈环节。

2.1.3　神经网络的仿真

神经网络的仿真过程实质上是神经网络根据网络输入数据，通过数值计算得出相应网络输出的过程。通过仿真，我们可以及时了解当前神经网络的性能，从而决定是否对网络进行进一步的训练。根据网络输入数据形式和神经网络的不同类型，神经网络的仿真可以采取不同的形式。

对于静态神经网络，由于网络中不含有任何延迟或反馈环节，所以网络输入数据的先后顺序对仿真结果没有任何影响。因此，网络输入数据不论是采用矩阵形式（数据顺序无先后），还是序列形式（数据顺序有先后，在 MATLAB 中通常以单元数组的形式表示），静态网络的仿真结果都是相同的。

对于动态神经网络，由于网络中含有反馈或延迟环节，因而网络的仿真输出不仅与当前的网络输入数据有关，而且还与过去的输入数据有关，即与输入数据的先后顺序有关。





因此，当动态神经网络接受以序列形式描述的输入数据时，即使只是序列中数据元素的顺序不同，仿真结果也会截然不同。

2.1.4 神经网络的学习与训练

学习特性是神经网络的基本特性，神经网络的学习与训练是通过网络权值和阈值的调节来实现的。根据学习过程的组织和管理方式不同，学习算法可分为有监督学习和无监督学习两大类。

对于有监督学习（如图 2.6 所示），网络训练往往要基于一定数量的训练样例或样本，训练样本通常由输入矢量和目标矢量组成。在学习和训练过程中，神经网络不断地将其实际输出与目标输出进行比较，并根据比较结果或误差，按照一定的规则或算法对网络权值和阈值进行调节，从而使网络的输出逐渐接近目标值。最典型的有监督学习算法的代表是 BP（Back Propagation）算法——误差反向传播算法。

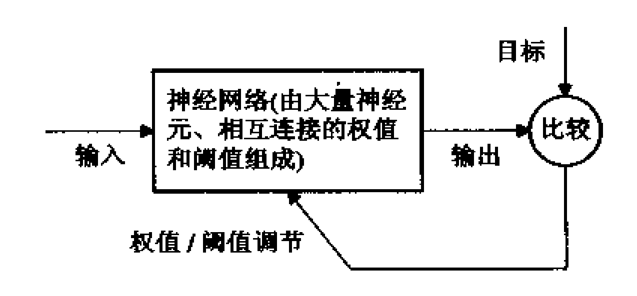


图 2.6 有监督学习原理图

无监督学习是一种自组织学习，即网络的学习过程完全是一种自我学习的过程，不需要提供学习样本或外界反馈。在学习过程中，网络只需响应输入信号的激励，按照某种规则反复调节网络权值和阈值，直到最后形成某种有序的状态。例如，在很多情况下，无监督学习算法可以用来做聚类分析，即通过学习将输入模式划分为有限的模式类别。无监督学习算法的典型代表是 Hebb 学习律。

根据每次网络训练任务量的不同，神经网络的训练方式可分为渐进式训练和批量式训练。渐进式训练（Incremental Training）是一种在线学习方式，即神经网络每接收一对输入矢量和目标矢量，便对网络权值和阈值进行适时的调整；而批量式训练（Batch Training）则是在所有的输入矢量和目标矢量集准备完成之后才开始根据相应的学习算法对网络权值和阈值进行批量调整。渐进式训练和批量式训练既适用于动态网络，也适用于静态网络。同一神经网络，采用两种不同的训练方式得出的训练结果是不同的。

在 MATLAB 6.x 中，对神经网络的训练统一由 train 函数和 adapt 函数来完成。train 函数只能对网络进行批量式训练（若训练数据为序列形式，则在训练前 train 函数会自动将其转换为矩阵或矢量形式），它是通过调用在生成网络对象 net 时所定义的训练算法（net.trainFcn）来完成不同类型网络的训练。adapt 函数一般用于渐进式训练，也可以用于批量式训练，当训练样本数据采用矩阵或矢量形式时，进行批量式训练；当训练样本数据采用序列形式时，则进行渐进式训练。adapt 函数是通过调用在生成网络对象 net 时所定义的训练算法（net.adaptFcn）来完成不同类型网络的训练。

在神经网络学习和训练过程中，选用何种训练方式，采用何种训练函数，应该根据具体的网络形式和具体问题的类型与要求而定。



2.2　感知器神经网络

感知器（Perceptron）是由美国学者 F. Rosenblatt 于 1957 年在 M-P 模型和 Hebb 学习律的基础上提出来的，它是一个具有单层计算神经元的神经网络，网络的传递函数是线性阈值单元。原始的感知器神经网络只具有一个神经元，主要被用来模拟人脑的感知特征，由于采用阈值单元作为传递函数，所以感知器神经元只能输出两个值，即只具有两个状态。感知器特别适用于简单的模式分类问题。当它用于两类模式的分类时，相当于在高维样本空间，用一个超平面将两类样本分开。F. Rosenblatt 已经证明，如果两类模式是线性可分的（指存在一个超平面将它们分开），则算法一定是收敛的。但是，单层感知器网络只能用来解决线性可分问题，而对于非线性或线性不可分问题则无能为力。

2.2.1　感知器神经网络结构

1. 感知器神经元模型

图 2.7 描述了一个由阈值函数（hardlim）运算单元组成的感知器神经元。采用阈值函数作为神经元的传递函数是感知器神经元的典型特征。hardlim 的函数形式参见表 2-1。可见，当 $\boldsymbol{wp}+b\geq 0$ 时，神经元输出 $a=1$；当 $\boldsymbol{wp}+b<0$ 时，神经元输出 $a=0$。显然，$\boldsymbol{wp}+b$ 构成了感知器神经元输出的决策边界。

正是基于阈值函数的二值特性，使得感知器神经元可以将输入 N 维空间划分为两个互不重叠的区域。对于一个具有二维输入的感知器神经元，当权值 $w_{1,1}=1$，$w_{1,2}=2$，阈值 $b=-2$ 时，它对输入二维平面的划分情况如图 2.8 所示。

由上述神经元确定的决策边界方程为

$$\boldsymbol{wp}+b=[1,2]\boldsymbol{p}+(-2)=p_1+2p_2-2=0$$

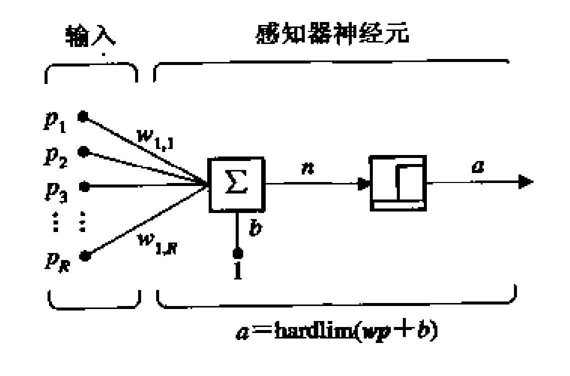


图 2.7　感知器神经元模型

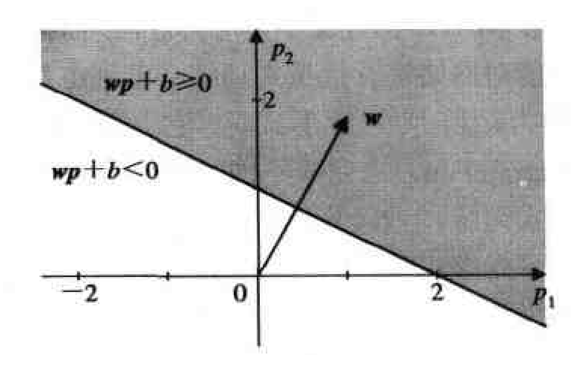
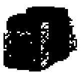


图 2.8　感知器神经元对二维平面的划分情况



2. 感知器神经网络结构

单层感知器神经网络结构如图 2.9 所示。网络具有 R 维输入，通过权值矩阵 $IW^{1,1}(S^1\times R)$ 与 S^1 个感知器神经元相连，感知器的输出为

$$a^1=\text{hardlim}(IW^{1,1}p+b^1)$$

由图 2.9 可见，感知器神经网络只有一层神经元，这是由感知器学习规则所决定的，因为感知器学习规则只能训练单层神经网络。感知器神经网络这种结构上的局限性也在一定程度上限制了其应用范围。

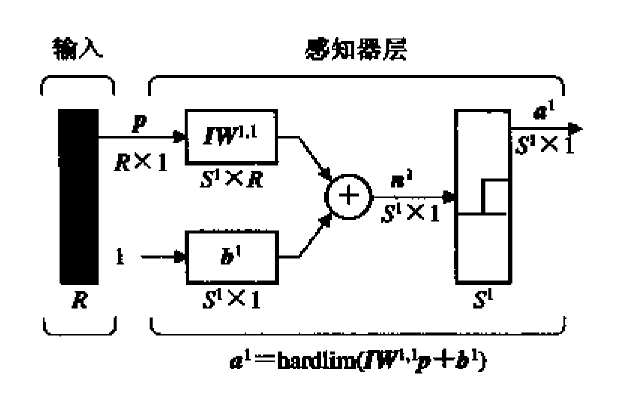


图 2.9　感知器神经网络结构图

2.2.2　感知器神经网络的设计

1. 感知器神经网络的生成及初始化

函数 newp 可以用来生成一个感知器神经网络，其调用格式为

```
net = newp ( PR , S )
```

其中，PR 为 $R\times2$ 维矩阵，表示 R 维输入矢量中每维输入的最小值和最大值之间的范围；S 表示感知器网络中神经元的个数；函数返回值 net 代表生成的感知器神经网络对象。在 MATLAB 中，net 是以结构体的形式加以存储的，神经网络对象的详细结构可参见 3.1 节。下面所给代码表示生成一个具有二维输入、两个神经元的感知器网络。

```
net = newp ( [-2, +2 ; -2, +2 ], 2 );
W = net.IW {1, 1}

W =
     0     0
     0     0
b = net.b {1}
b =
     0
     0
```

可见，示例中生成的感知器网络初始权值和阈值均为零，这是因为在生成网络对象时，newp 已使用默认的初始化函数 initzero 对网络的权值和阈值进行了零初始化。用户可以改变初始化函数的形式，并使用 init 函数对网络重新进行初始化，例如：

```
net.inputweights{1, 1}. InitFcn = 'rands';
net.biases{1}.InitFcn = 'rands';
net = init(net);
```



```
W = net.IW{1, 1}
W =
        0.2137      0.7826
       -0.0280      0.5242
b = net.b{1}
b =
        0.9003
       -0.5377
```

2. 感知器神经网络的学习规则

函数 learnp 是在感知器神经网络学习过程中计算网络权值和阈值修正量最基本的规则函数，该学习规则的基本原理为

权值增量：$\Delta \boldsymbol{W} = (\boldsymbol{t} - \boldsymbol{a})\boldsymbol{p}^{\mathrm{T}} = \boldsymbol{e}\boldsymbol{p}^{\mathrm{T}}$

阈值增量：$\Delta \boldsymbol{b} = (\boldsymbol{t} - \boldsymbol{a})(1) = \boldsymbol{e}$

权值更新：$\boldsymbol{W}^{\mathrm{new}} = \boldsymbol{W}^{\mathrm{old}} + \Delta \boldsymbol{W}$

阈值更新：$\boldsymbol{b}^{\mathrm{new}} = \boldsymbol{b}^{\mathrm{old}} + \Delta \boldsymbol{b}$

其中，$\boldsymbol{p}$ 为输入矢量，学习误差 $\boldsymbol{e}$ 为目标矢量 $\boldsymbol{t}$ 和网络实际输出矢量 $\boldsymbol{a}$ 之间的差值。上述原理对应的 MATLAB 语句为

```
e = t – a;
dW = learnp ( W, p, [ ], [ ], [ ], [ ], e, [ ], [ ], [ ] );
db = e;
W = W + dW;
b = b + db;
```

由上述原理不难发现，learnp 函数的学习规则要受输入矢量 $\boldsymbol{p}$ 大小变化的影响。当输入样本矢量中含有奇异样本点时，往往会导致感知器神经网络的学习训练时间加长，因此为了消除训练时间对奇异样本的敏感性，可以对上述算法作如下改进：

$$\Delta \boldsymbol{W} = (\boldsymbol{t} - \boldsymbol{a})\boldsymbol{p}^{\mathrm{T}} / \|\boldsymbol{p}\| = \boldsymbol{e}\boldsymbol{p}^{\mathrm{T}} / \|\boldsymbol{p}\|$$

显然，改进后的算法，即归一化感知器学习规则对输入样本矢量的大小变化不敏感。感知器归一化学习规则可以通过调用函数 learnpn 加以实现。

3. 感知器神经网络的训练与仿真

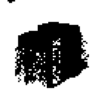
感知器神经网络的训练可以通过反复调用 adapt 函数或直接调用 train 函数来完成。采用 adapt 函数对感知器网络进行训练的格式为

```
[net, y, e] = adapt (net, p, t )
```

其中，net 为训练前、后的网络对象，y 和 e 分别为训练后网络的输出和误差。

采用 train 函数可以以批量训练方式对感知器网络进行训练，训练速度较快，但不能保证每次训练结果都能达到要求。train 函数的常用调用格式为

```
net = train (net, p, t)
```



对于感知器网络的训练，建议采用 adapt 函数，因为已经证明，对于任何一个线性可分问题，利用 adapt 函数经过有限步训练均可收敛，但对于 train 函数则不能保证这一点。

利用 sim 函数可以对感知器神经网络进行仿真，从而检验网络的输出和训练结果。sim 函数的常用格式为

```
a = sim (net, p)
```

值得注意的是，在经过足够多的训练和仿真之后，如果网络的性能还是达不到要求，则应仔细分析一下，即感知器神经网络是否适合解决当前问题。

2.2.3　感知器神经网络的局限性

由于感知器神经网络在结构和学习规则上的局限性，使得其应用也受到一定的限制。首先，由于感知器网络的神经元传递函数采用 hardlim 函数，所以只能输出 0 或 1 两个值；其次，单层感知器神经网络只能解决线性可分问题，即只能对线性可分的输入样本矢量进行分类，而对于线性不可分问题则无能为力。例如，对于如图 2.10 所示的两类点集，仅采用单层感知器神经网络是不可能找到一条决策边界将两类点集分开的。

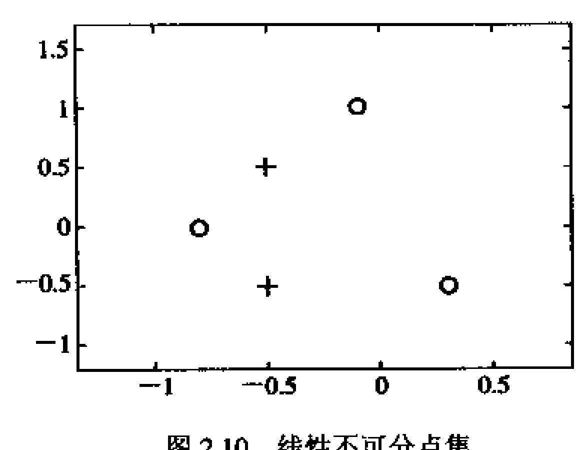
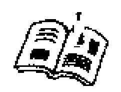


图 2.10　线性不可分点集

解决线性不可分问题的途径有两种：一是采用多层感知器网络；二是采用功能更强大的神经网络，如 BP 网络等，这将在以后各章节中加以讨论。

2.3　线性神经网络

线性神经网络是最简单的一种神经网络，它由一个或多个线性神经元构成。1960 年由 B. Widrow 和 M. E. Hoff 提出的自适应线性单元（Adaline）网络是线性神经网络最早的典型代表。线性神经网络采用线性函数作为传递函数，因此其输出可以取任意值。线性神经网络可以采用基于最小二乘算法（LMS）的 Widrow-Hoff 学习规则来调节网络的权值和阈值，其收敛速度和精度都有较大的改进。此外，采用 newlind 函数还可以直接根据网络的输入矢量和目标矢量设计出期望的线性网络。和感知器神经网络一样，线性神经网络只能反映输入和输出样本矢量间的线性映射关系，它也只能解决线性可分问题。线性神经网络在函数拟合、信号滤波、预测和控制等方面都有着广泛的应用。



2.3.1 线性神经网络结构

1. 线性神经元模型

线性神经元模型的结构如图2.11所示。

与感知器神经元不同的是，线性神经元采用的传递函数为线性函数 purelin，其输入与输出之间是简单的纯比例关系。purelin 函数的具体形式参见表 2-1。线性神经元的输出可以取任意值，其输入、输出关系为

$$a = \text{purelin}(wp + b) = wp + b$$

2. 线性神经网络结构

图2.12给出了具有 R 维输入的单层（包含 S 个神经元）线性神经网络模型。

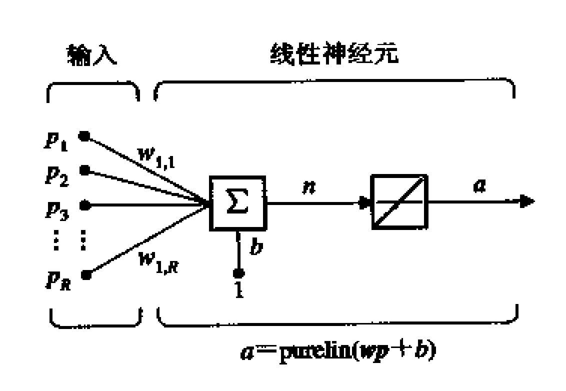


图2.11　线性神经元模型

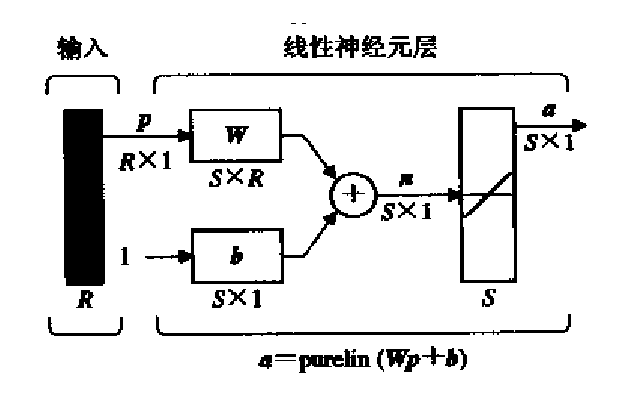


图2.12　线性神经网络

2.3.2 线性神经网络设计

1. 线性神经网络的生成及初始化

利用函数 newlin 可以生成一个线性神经网络。其常用的格式为

```
net = newlin ( PR , S )
```

其中，PR 为 $R\times2$ 维矩阵，表示 R 维输入矢量中每维输入的最小值和最大值之间的范围；S 表示线性神经网络的输出个数，即网络层神经元的个数；函数返回变量 net 为生成的线性神经网络对象。在线性网络生成的同时，newlin 已调用默认的初始化函数 initzero 对网络权值和阈值进行了零初始化。和感知器神经网络的初始化一样，用户可以任意改变线性网络权值和阈值的初始化函数，并使用 init 函数对网络重新进行初始化。

2. 线性神经网络的学习规则

线性神经网络权值和阈值的学习规则采用的是基于最小二乘原理的 Widrow-Hoff 学习算法。基于 Widrow-Hoff 学习算法的权值和阈值调节原理如下：

$$\boldsymbol{W}(k+1) = \boldsymbol{W}(k) + \Delta\boldsymbol{W}(k),\quad \boldsymbol{b}(k+1) = \boldsymbol{b}(k) + \Delta\boldsymbol{b}(k)$$



其中

$$\Delta W(k) = -\alpha\frac{\partial e^2(k)}{\partial W} \qquad \Delta b(k) = -\alpha\frac{\partial e^2(k)}{\partial b}$$

由于

$$\frac{\partial e^2(k)}{\partial W} = 2e(k)\frac{\partial e}{\partial W} \qquad \frac{\partial e^2(k)}{\partial b} = 2e(k)\frac{\partial e}{\partial b}$$

且有

$$\frac{\partial e}{\partial W} = \frac{\partial}{\partial W}\left[t(k) - (W * p(k) + b)\right] = -p(k) \qquad \frac{\partial e}{\partial b} = -1$$

所以，上述权值和阈值的调节公式可变为

$$W(k+1) = W(k) + 2\alpha e(k)p^{\mathrm{T}}(k) \qquad b(k+1) = b(k) + 2\alpha e(k)$$

即

$$W(k+1) = W(k) + \eta e(k)p^{\mathrm{T}}(k) \qquad b(k+1) = b(k) + \eta e(k)$$

其中，η 为学习速率。当 η 较大时，学习速率加快（η 取值过大时有可能使学习变得不稳定），反之亦然。

在 MATLAB 中，Widrow-Hoff 学习算法对应的学习函数为 learnwh。

3. 线性神经网络的训练和仿真

对线性神经网络的训练可以调用 train 函数完成。利用 train 函数对线性神经网络进行训练实际上是根据所给的“输入一目标”样本矢量集，调用神经网络生成时所定义的权值和阈值学习函数 learnwh 对网络不断进行调节，最终使网络输出接近目标输出的过程。下面的代码给出了一个典型的具有二维输入线性神经元的训练过程。

```
P = [1, 2, -1, 0 ; 2, 2, 0, -1 ];
t = [0, 0, 1, 1];
net = newlin ( [-2, 2 ; -2, 2 ], 1 );
net.trainParam.goal = 0.01;
[net, tr] = train ( net, P, t );
```

训练中会显示如下信息：

```
TRAINB, Epoch 0/100, MSE 0.5/0.01.
TRAINB, Epoch 25/100, MSE 0.121667/0.01.
TRAINB, Epoch 50/100, MSE 0.0334739/0.01.
TRAINB, Epoch 75/100, MSE 0.012111/0.01.
TRAINB, Epoch 82/100, MSE 0.00983141/0.01.
TRAINB, Performance goal met.
```

可见，当训练到第 82 步时，网络性能达标。此时的权值和阈值矩阵为



```
W = net.IW{1,1}
W =
    -0.1854    -0.2112
b = net.b{1}
b =
    0.6973
```

利用 sim 等相关函数可以对训练好的线性神经网络进行仿真和误差分析：

```
A = sim ( net, P )
A =
    0.0894    -0.0959    0.8827    0.9086
error = t - A
error =
    -0.0894    0.0959    0.1173    0.0914
mse = mse ( error )
mse =
    0.0098
```

4. 线性神经网络的直接设计法

与其他神经网络设计不同的是，线性神经网络可以根据输入和目标矢量直接设计出来。函数 newlind 无须经过训练，就可以直接设计出线性神经网络，使得网络实际输出与目标输出的平方和误差 SSE 为最小，其常用格式为

```
net = newlind ( P, T )
```

2.3.3 自适应滤波线性神经网络

1. 自适应滤波线性神经网络结构

自适应滤波是线性神经网络一个很重要的应用。自适应滤波线性神经网络（简称自适应滤波网络）是线性神经网络的一种特殊形式，它与一般线性神经网络的不同之处在于自适应滤波网络引入了如图 2.13 所示的延迟链 TDL。

可见，自适应滤波网络的输入是由同一输入信号的若干延迟信号所构成的，即有

$$pd_1(k) = p(k)$$

$$pd_2(k) = p(k-1)$$

$$\vdots$$

$$pd_N(k) = p(k-N+1)$$

一个典型的自适应滤波网络的结构如图 2.14 所示，其中，线性层只有一个神经元。该神经网络的输出为





$$a(k) = \text{purelin}(\boldsymbol{wp} + b) = \sum_{i=1}^{N} w_{1,i}\, p(k - i + 1) + b$$

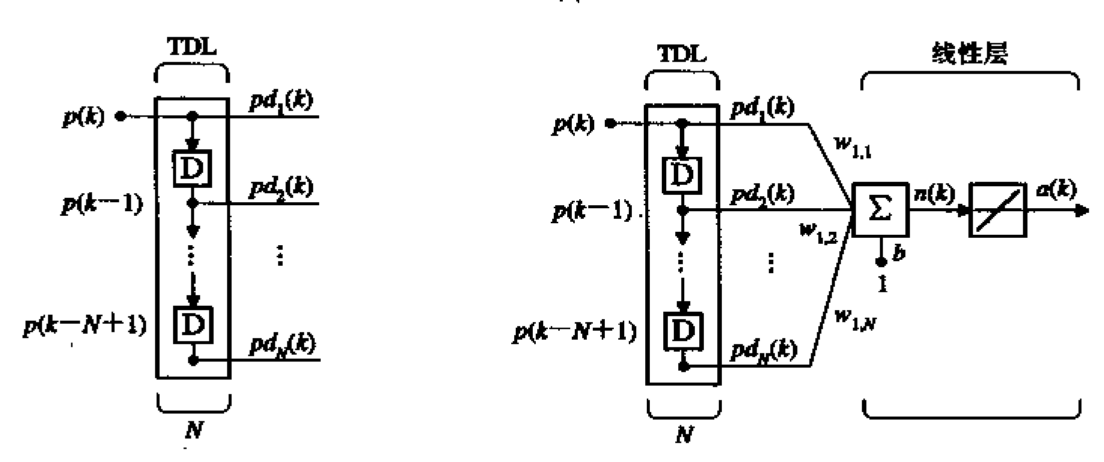


图 2.13　TDL 延迟链　　　　图 2.14　自适应滤波网络结构

可见，自适应滤波网络的输出是由输入信号 $p(k)$ 及其延迟信号的线性组合构成的，这正好与数字信号处理领域所谓的有限冲击响应（FIR）滤波器的结构形式相吻合。

当然，自适应滤波网络的线性层也可以由多个线性神经元组成。一个典型的具有多个神经元的自适应滤波网络结构如图 2.15 所示。图 2.16 给出了其缩略形式描述图。

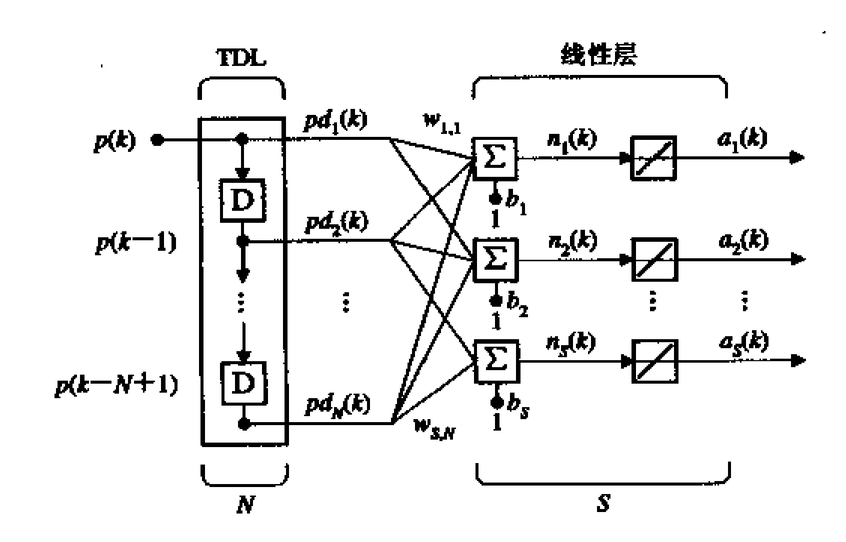


图 2.15　具有 S 个神经元的自适应滤波网络

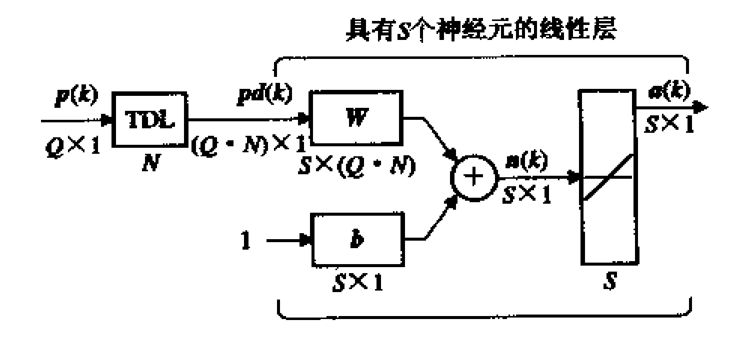


图 2.16　具有 S 个神经元的自适应滤波网络的缩略形式



迄今为止，自适应滤波网络已成为应用最为广泛的神经网络之一，它在信号滤波、预测与控制等领域都有着广泛的应用。本书第四章给出了关于自适应滤波网络的几个典型应用实例。

2. 自适应滤波线性神经网络设计

自适应滤波网络的生成可以采用两种方式：一种是通过调用 newlin 函数直接生成带有延迟链的自适应滤波网络；另一种则是首先利用 newlin 函数生成不带延迟链的线性网络，然后通过网络重定义将延迟链加入预生成的线性网络中。图 2.17 为一个单神经元自适应滤波网络的结构示意图。

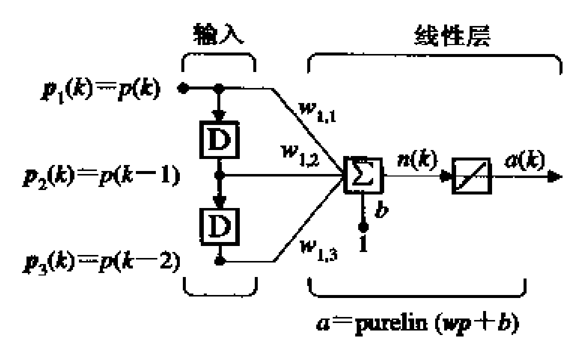


图 2.17　单神经元自适应滤波网络

该自适应滤波网络可以采用如下两种方式来生成（假设网络输入范围为[0，10]）：

方式一：

```
net = newlin ( [0,10] , 1 , [ 0 1 2 ] );
```

方式二：

```
net = newlin ( [0,10] , 1 );
net.inputWeights{1,1}.delays = [0 1 2];
```

其中，[0 1 2]中各元素分别表示自适应滤波网络各维输入所对应的延迟量。下面我们对所生成的自适应滤波网络分别进行初始化、仿真和训练。

自适应滤波网络的初始化与一般的线性神经网络基本相同，只是在初始化网络权值和阈值的同时，自适应滤波网络还要对延迟输入的初始值进行设置。

首先，对网络初始权值和阈值进行设置：

```
net.IW{1,1} = [ 7 , 8 , 9 ];
net.b{1} = 0;
```

其次，在对网络进行仿真之前需要对延迟输入 $p_2(k)$ 和 $p_3(k)$ 的初始值进行设置：

```
pi = { 1 , 3 };
```

即对应 $p_2(0)=1$、$p_3(0)=3$。

定义输入矢量如下：

```
P = { 3 , 4 , 5 , 6 };
```

现在可以调用 sim 函数对上面所创建的自适应滤波网络进行仿真：

```
[ a , pf ] = sim ( net , P , pi );
a =
      [54]    [79]    [94]    [118]
pf =
      [5]    [6]
```

其中，pf 为仿真后延迟输入的终值。有兴趣的读者可以利用笔算对上述网络的仿真结果加以验证，以便能更好地理解自适应滤波网络的工作原理。



下面，利用 adapt 函数对自适应滤波网络进行训练，使自适应滤波网络能够根据输入序列 P 输出期望的目标序列 T。

```
T = { 10 , 20 , 30 , 40 };
net.adaptParam.passes = 500;
[ net , y , e ] = adapt ( net , P , T , pi );
```

经过训练，网络的实际输出响应为

```
y =
    [11.2888]    [22.9083]    [27.5403]    [39.4193]
```

可见，训练结果基本满意，要想得到更高的匹配精度，还需要继续对网络进行训练。

此外，还可以采用 newlind 函数对上述自适应滤波网直接进行设计，即

```
net = newlind ( P , T , pi );
y = sim ( net , P )
y =
    [10.0000]    [20.0000]    [30.0000]    [40.0000]
```

可见，利用 newlind 函数直接设计出的自适应滤波网络实现了零误差输出。

2.3.4　线性神经网络的局限性

线性神经网络只能反映输入和输出样本矢量间的线性映射关系，和感知器神经网络一样，它也只能解决线性可分问题。由于线性神经网络的误差曲面是一个多维抛物面，所以在学习速率足够小的情况下，对于基于最小二乘梯度下降原理进行训练的线性神经网络总可以找到一个最优解。但是，尽管如此，对线性神经网络的训练并不一定总能达到零误差。线性神经网络的训练性能要受网络规模和训练样本集大小的限制。若线性神经网络的自由度（即神经网络所有权值和阈值的个数总和）小于训练样本集中“输入—目标”矢量的对数，且各样本矢量与线性无关，则网络训练不可能达到零误差，而只能得到一个使网络误差最小的解；反之，若网络自由度大于样本集的个数，则会得到无穷多个使网络训练误差为零的解。此外，值得注意的是，线性神经网络的训练和性能要受到学习速率参数的影响，过大的学习速率可能会导致网络性能发散。

2.4　BP 神经网络

BP 神经网络通常是指基于误差反向传播算法（BP 算法）的多层前向神经网络，它是 D. E. Rumelhart 和 J. L. McCelland 及其研究小组在 1986 年研究并设计出来的。BP 算法已成为目前应用最为广泛的神经网络学习算法，据统计有近 90%的神经网络应用是基于 BP 算法的。与感知器和线性神经网络不同的是，BP 网络的神经元采用的传递函数通常是 Sigmoid 型可微函数，所以可以实现输入和输出间的任意非线性映射，这使得它在诸如函数逼近、模式识别、数据压缩等领域有着更加广泛的应用。



2.4.1 BP 神经网络结构

1. BP 神经元模型

图 2.18 给出了一个基本的 BP 神经元模型，与其他神经元模型不同的是，BP 神经元模型中的传递函数 f 通常取可微的单调递增函数，如对数 Sigmoid 函数 logsig、正切 Sigmoid 函数 tansig 和线性函数 purelin 等。

BP 网络最后一层神经元的特性决定了整个神经网络的输出特性。当最后一层神经元采用 Sigmoid 型函数时，那么整个网络的输出就被限制在一个较小的范围内；如果最后一层神经元采用 purelin 型函数，则整个网络输出可以取任意值。

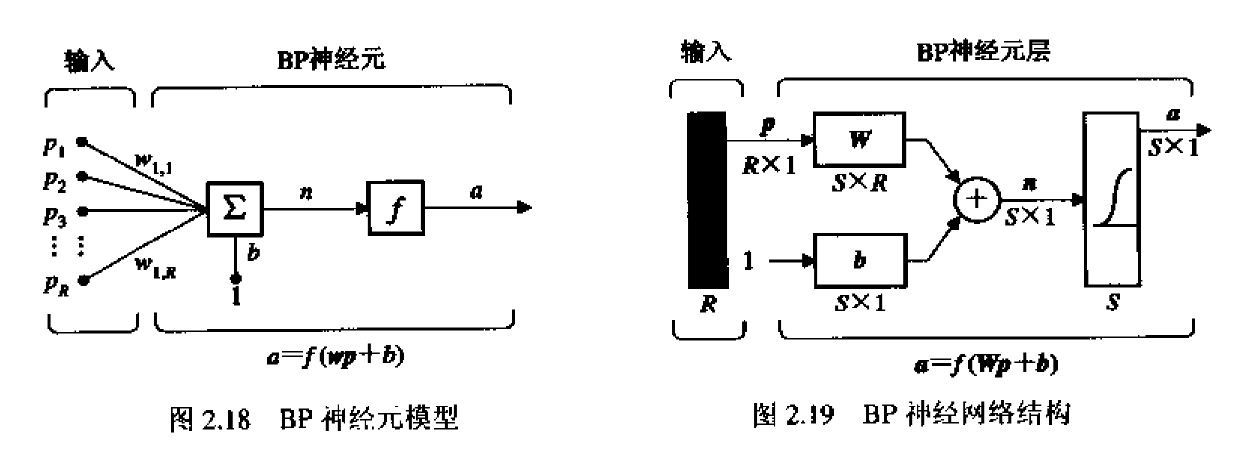


图 2.18　BP 神经元模型　　　　图 2.19　BP 神经网络结构

2. BP 神经网络结构

BP 神经网络通常采用基于 BP 神经元的多层前向神经网络的结构形式。一个典型的 BP 网络结构如图 2.19 所示。

BP 神经网络通常具有一个或多个隐层，其中，隐层神经元通常采用 Sigmoid 型传递函数，而输出层神经元则采用 purelin 型传递函数。图 2.20 给出了一个具有单隐层的 BP 神经网络模型，其中，输入维数为 2，隐层含有 4 个神经元，采用 tansig 作为传递函数，输出层包含 3 个神经元，采用 purelin 传递函数。理论已经证明，具有如图 2.20 所示结构的 BP 神经网络，当隐层神经元数目足够多时，可以以任意精度逼近任何一个具有有限间断点的非线性函数。

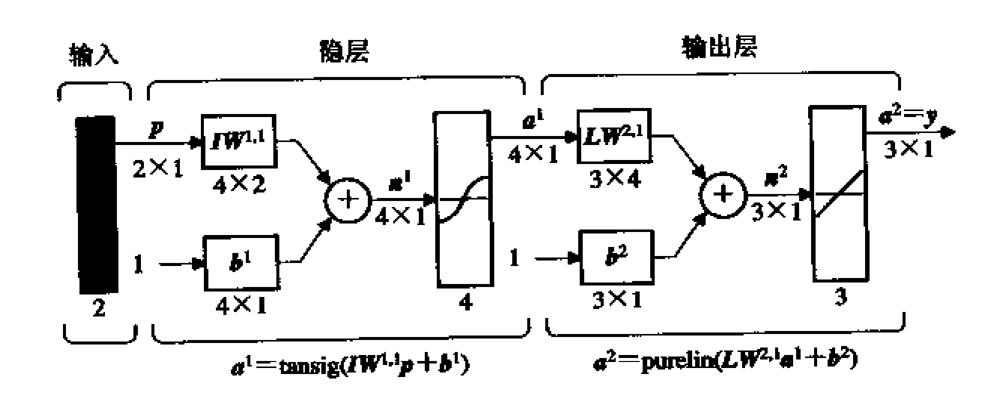


图 2.20　具有单隐层的 BP 神经网络



2.4.2 BP神经网络设计

1. BP神经网络的生成及初始化

采用newff函数可以用来生成BP网络。newff函数的常用格式为

net = newff (PR , [S1 S2 ··· SN] , {TF1 TF2 ··· TFN } , BTF)

其中，PR为$R\times 2$维矩阵，表示R维输入矢量中每维输入的最小值与最大值之间的范围；若神经网络具有N层，则[S1 S2 ··· SN]中各元素分别表示各层神经元的数目；{TF1 TF2 ··· TFN }中各元素分别表示各层神经元采用的传递函数；BTF表示神经网络训练时所使用的训练函数。例如，下面代码表示生成一个两层BP神经网络，其中，输入维数为2，各维输入的取值范围分别为[0 , 10]和[-1 , 2]；输入层和输出层神经元的个数分别为5和1，各层神经元的传递函数分别取tansig函数和purelin函数，BP网络的训练函数取trainlm：

```
net = newff ( [ 0 , 10 ; -1 , 2 ] , [ 5 , 1 ] , { 'tansig' , 'purelin' } , 'trainlm' );
```

net为生成的BP网络对象。newff在生成BP网络的同时即对网络各层的权值和阈值自动进行了初始化，根据不同的需求，用户可以对各层网络权值或阈值的初始化函数重新定义，并使用init函数重新对网络进行初始化。

2. BP神经网络的学习规则

BP神经网络的学习规则，即权值和阈值的调节规则采用的是误差反向传播算法（BP算法）。BP算法实际上是Widrow-Hoff算法在多层前向神经网络中的推广。和Widrow-Hoff算法类似，在BP算法中，网络的权值和阈值通常是沿着网络误差变化的负梯度方向进行调节的，最终使网络误差达到极小值或最小值，即在这一点误差梯度为零。

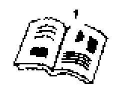
限于梯度下降算法的固有缺陷，标准的BP学习算法通常具有收敛速度慢、易陷入局部极小值等缺点，因此出现了许多改进的算法，我们将在下一节对这些算法进行介绍。

3. BP神经网络的训练和仿真

在BP神经网络生成和初始化以后，即可利用现有的“输入—目标”样本矢量数据对网络进行训练。BP网络的训练通常采用train函数来完成。针对不同的问题，在训练之前有必要对网络的训练参数net.trainParam进行适当的设置。表2-2列出了网络对象的一些主要训练参数及含义。

表2-2　几个主要的神经网络训练参数及含义

训练参数	参数含义	默认值
net.trainParam.epochs	训练步数	100
net.trainParam.show	显示训练结果的间隔步数	25
net.trainParam.goal	训练目标误差	0
net.trainParam.time	训练允许时间	Inf
net.train.Param.min_grad	训练中最小允许梯度值	1e-6



在设置完训练参数之后，就可以调用 train 函数对 BP 网络进行训练了。train 函数的常用格式如下：

[net , tr] = train (net , P , T)

其中，P 为输入样本矢量集；T 为对应的目标样本矢量集；等号右、左两侧的 net 分别用于表示训练前、后的神经网络对象；tr 存储训练过程中的步数信息和误差信息。训练过程中，训练函数会根据设定的 net.trainParam.show 值自动显示当前训练结果信息，并给出网络误差实时变化曲线。当训练步数大于 net.trainParam.epochs、训练误差小于 net.trainParam.goal、训练时间超过 net.trainParam.time，或误差梯度值小于 net.trainParam.min_grad 时，训练都将被自动终止，并返回训练后的神经网络对象。

为了提高神经网络的训练效率，在某些情况下需要对"输入—目标"样本集数据作必要的预处理。如利用 premnmx 或 prestd 函数可以对输入和目标数据集进行归一化处理，使其落入[-1,1]区间；利用 prepca 函数可以对输入样本集进行主元分析，以减小输入各样本矢量间的相关性，从而起到降维的目的。有关函数的详细使用方法参见 3.2.19 小节。

利用 sim 函数可以对训练后的网络进行仿真。此外，神经网络工具箱还提供了 postreg 函数，该函数可对训练后网络的实际输出（仿真输出）和目标输出作线性回归分析，以检验神经网络的训练效果。

2.4.3　BP 神经网络的快速学习算法与选择

为了克服常规 BP 学习算法的缺陷，MATLAB 神经网络工具箱对常规 BP 算法进行了改进，并提供了一系列快速学习算法，以满足解决不同问题的需要。

快速 BP 算法从改进途径上可分为两大类：一类是采用启发式学习方法，如引入动量因子的学习算法（traingdm 函数）、变学习速率学习算法（traingda 函数）和"弹性"学习算法（trainrp 函数）等；另一类则是采用更有效的数值优化方法，如共轭梯度学习算法（包括 traincgf、traincgp、traincgb、trainscg 等函数）、quasi-Newton 算法（包括 trainbfg、trainoss 等函数）以及 Levenberg-Marquardt 优化方法（trainlm 函数）等。

对于不同的问题，在选择学习算法对 BP 网络进行训练时，不仅要考虑算法本身的性能，还要视问题的复杂度、样本集大小、网络规模、网络误差目标和所要解决的问题类型（判断其是属于"函数拟合"还是"模式分类"问题）而定。表 2-3 对几种典型的快速学习算法进行了比较，以作为选择学习算法的参考。

表 2-3　几种典型的快速学习算法性能的比较

学习算法	适用问题类型	收敛性能	占用存储空间	其他特点
trainlm	函数拟合	收敛快，收敛误差小	大	性能随网络规模增大而变差
trainrp	模式分类	收敛最快	较小	性能随网络训练误差减小而变差



续表

trainscg	函数拟合、模式分类	收敛较快，性能稳定	中等	尤其适用于网络规模较大的情况
trainbfg	函数拟合	收敛较快	较大	计算量随网络规模的增大呈几何增长
traingdx	模式分类	收敛较慢	较小	适用于"提前停止"方法，可提高网络的推广能力

2.4.4 BP 神经网络推广能力的提高

推广能力（Generalization）是衡量神经网络性能好坏的重要标志。一个"过度训练"（Overfitting）的神经网络可能会对训练样本集达到较高的匹配效果，但对于一个新的输入样本矢量却可能会产生与目标矢量差别较大的输出，即神经网络不具有或具有较差的推广能力。MATLAB 神经网络工具箱给出了两种用于提高神经网络推广能力的方法，即正则化方法（Regularization）和提前停止（Early stopping）方法，下面分别对这两种方法作一介绍。

1. 正则化方法

在训练样本集大小一定的情况下，网络的推广能力与网络的规模直接相关。如果神经网络的规模远远小于训练样本集的大小，则发生"过度训练"的机会就很小，但是对于特定的问题，确定合适的网络规模（通常指隐层神经元数目）往往是一件十分困难的事情。正则化方法是通过修正神经网络的训练性能函数来提高其推广能力的。一般情况下，神经网络的训练性能函数采用均方误差函数 mse，即

$$\mathrm{mse}=\frac{1}{N}\sum_{i=1}^{N}(e_i)^2=\frac{1}{N}\sum_{i=1}^{N}(t_i-a_i)^2$$

在正则化方法中，网络性能函数经改进变为如下形式：

$$\mathrm{msereg}=\gamma\cdot\mathrm{mse}+(1-\gamma)\mathrm{msw}$$

其中，γ 为比例系数，msw 为所有网络权值平方和的平均值，即

$$\mathrm{msw}=\frac{1}{n}\sum_{j=1}^{N}w_j^2$$

可见，通过采用新的性能指标函数，可以在保证网络训练误差尽可能小的情况下使网络具有较小的权值，即使得网络的有效权值尽可能少，这实际上相当于自动缩小了网络的规模。

常规的正则化方法通常很难确定比例系数 γ 的大小，而贝叶斯正则化方法则可以在网络训练过程中自适应地调节 γ 的大小，并使其达到最优。在 MATLAB 工具箱中，贝叶斯正则化方法是通过 trainbr 函数来实现的。实践证明，采用 trainbr 函数训练后的 BP 网络具有较好的推广能力，但值得注意的是，该算法只适用于小规模网络的函数拟合或逼近问





题，不适用于解决模式分类问题，而且其收敛速度一般比较慢。采用贝叶斯正则化方法提高网络推广能力的实例参见 4.3 小节的例 4.13。

2. 提前停止

“提前停止”是提高神经网络推广能力的另一种有效方法。在该方法中，训练样本集在训练之前需要被划分为训练集、验证集或测试集，其中测试集可选。训练集用于对神经网络进行训练，验证集用于在神经网络训练的同时监控网络的训练进程。在训练初始阶段，验证集形成的验证误差通常会随着网络训练误差的减小而减小，但是当网络开始进入“过度训练”时，验证误差就会逐渐增大，当验证误差增大到一定程度时，网络训练会提前停止，这时训练函数会返回当验证误差取最小值时的网络对象。测试集形成的测试误差在网络训练时未被使用，但它可以用来评价网络训练结果和样本集划分的合理性。若测试误差与验证误差分别达到最小值时的训练步数差别很大，或两者曲线的变化趋势差别较大，则说明样本集的划分不是很合理，需要重新划分。

“提前停止”方法可以和任何一种 BP 算法结合起来使用，其缺点是需要对样本集进行划分，且划分的合理性不易控制。采用了“提前停止”方法的 train 函数，其调用格式为

```
[ net , tr ] = train ( net , ptr , ttr, [ ] , [ ] , val , test );
```

或（当不提供测试集时）

```
[ net , tr ] = train ( net , ptr , ttr, [ ] , [ ] , val );
```

其中，ptr 和 ttr 分别代表训练集的输入矢量和目标矢量；val 为验证集，test 为测试集，val 和 test 通常采用结构体的形式存取相应样本集中的输入矢量和目标矢量，例如 val.P 和 val.T 可分别用以定义验证集的输入矢量和目标矢量。采用“提前停止”方法提高 BP 网络推广能力的实例可参见 4.3 小节的例 4.14。

2.4.5 BP 神经网络的局限性

BP 神经网络克服了感知器网络和线性神经网络的局限性，可以实现任意线性或非线性的函数映射。然而，由于 BP 神经网络是基于梯度下降的误差反向传播算法进行学习的，所以其网络训练速度通常很慢，而且很容易陷入局部极小点，尽管采用一些改进的快速学习算法可以较好地解决某些实际问题，但是在设计过程中往往都要经过反复的试凑和训练过程，无法严格保证每次训练时 BP 算法的收敛性和全局最优性。此外，BP 网络隐层神经元的作用机理及其个数选择已成为 BP 网络研究中的一个难点问题。

2.5 径向基函数网络

径向基函数（ RBF ）网络是以函数逼近理论为基础而构造的一类前向网络，这类网络的学习等价于在多维空间中寻找训练数据的最佳拟合平面。径向基函数网络的每个隐层神经元传递函数都构成了拟合平面的一个基函数，网络也由此而得名。径向基函数网络是一种局部逼近网络，即对于输入空间的某一个局部区域只存在少数的神经元用于决定网络



的输出。我们熟悉的 BP 网络则是典型的全局逼近网络，即对每一个输入/输出数据对，网络的所有参数均要调整。由于二者的构造本质不同，径向基函数网络与 BP 网络相比规模通常较大，但学习速度较快，并且网络的函数逼近能力、模式识别与分类能力都优于后者。

MATLAB 6.x 神经网络工具箱提供了径向基函数网络以及它的两种重要变型——广义回归网络和概率神经网络，其中，广义回归网络适用于解决函数逼近问题，概率神经网络多用来解决分类问题。工具箱函数 newrbe 和 newrb 可用于设计径向基函数网络，函数 newgrnn 和 newpnn 则分别用于广义回归网络和概率神经网络的设计。

2.5.1 径向基函数网络

1. 径向基函数神经元模型

一个具有 R 维输入的径向基函数神经元模型如图 2.21 所示。图中的‖dist‖模块表示求取输入矢量和权值矢量的距离。此模型中采用高斯函数 radbas 作为径向基神经元的传递函数，其输入 n 为输入矢量 p 和权值矢量 w 的距离乘以阈值 b。

高斯函数 radbas 是典型的径向基函数，其表达式为

$$f(x)=e^{-x^2}$$

其函数曲线如图 2.22 所示。

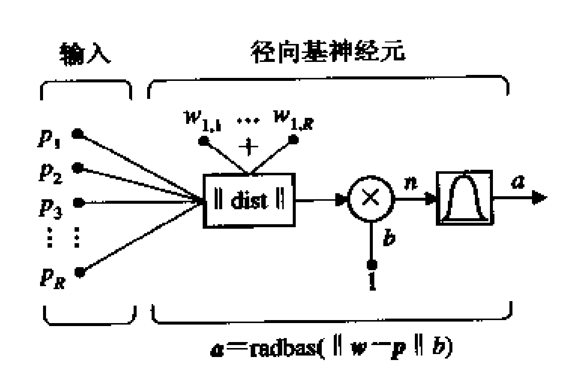


图 2.21　具有 R 维输入的径向基函数神经元

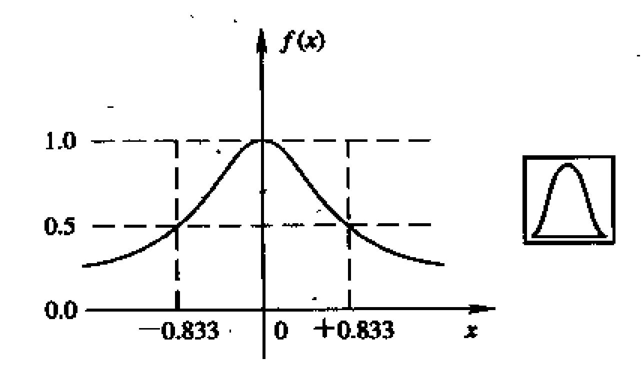


图 2.22　高斯径向基函数曲线

中心与宽度是径向基函数神经元的两个重要参数。神经元的权值矢量 w 确定了径向基函数的中心，当输入矢量 p 与 w 重合时，径向基函数神经元的输出达到最大值，当输入矢量 p 距离 w 越远时，神经元输出就越小。神经元的阈值 b 确定了径向基函数的宽度，当 b 越大，则输入矢量 p 在远离 w 时函数的衰减幅度就越大。

2. 径向基函数网络的结构

一个典型的径向基函数网络包括两层，即隐层和输出层。图 2.23 是一径向基函数网络的结构图。图中所示网络的输入维数为 R、隐层神经元个数为 S^1、输出个数为 S^2，隐层神经元采用高斯函数作为传递函数，输出层的传递函数为线性函数。图中 a_i^1 表示隐层



输出矢量 a^1 的第 i 个元素，w_i^1 表示第 i 个隐层神经元的权值矢量，即隐层神经元权值矩阵 W^1 的第 i 行。

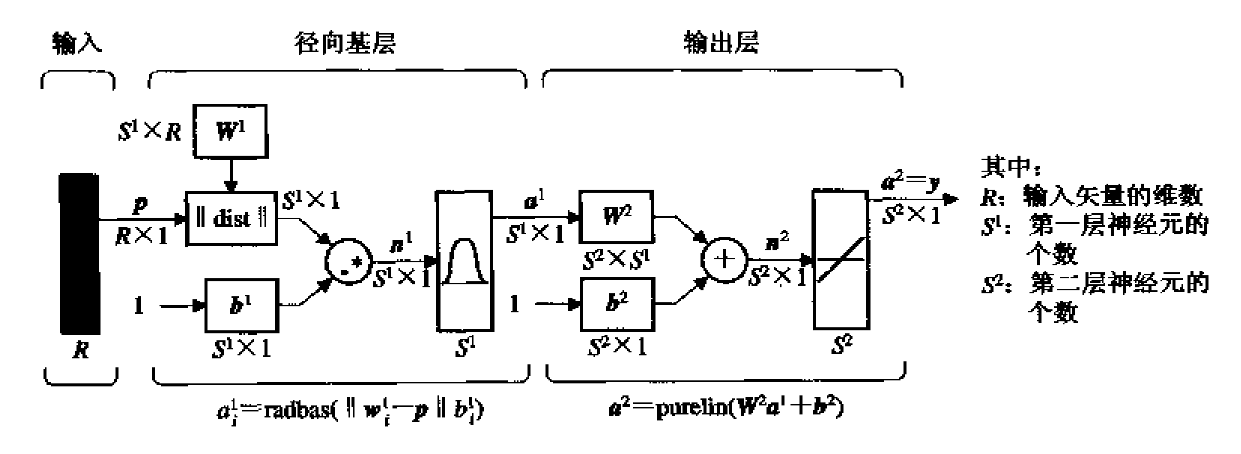


图 2.23 径向基函数网络结构图

3. 径向基函数网络的精确设计

函数 newrbe 可用来精确设计径向基函数网络，所谓精确，是指该函数生成的网络对于训练样本数据达到了零误差。函数 newrbe 的调用形式为

net = newrbe (P , T , SPREAD)

其中， P、T 分别为输入样本矢量集和输出目标矢量集构成的矩阵；SPREAD 是扩展常数，其缺省值为 1；函数返回值 net 为生成的网络对象，net 中的权值和阈值使得神经网络在输入为 P 时可以精确输出 T 。

函数 newrbe 在建立网络时生成的隐层神经元个数与矩阵 P 中的输入矢量个数相同，隐层神经元阈值取为 0.8632/ SPREAD，一般 SPREAD 的选取要足够大，以保证径向基神经元的响应在输入空间能够交迭。当输入矢量个数过多时，利用 newrbe 生成的网络会过于庞大，从而使网络实用性变差，因此在实际应用中使用更为广泛的径向基函数网络设计函数是 newrb 函数 。

4. 径向基函数网络的更有效设计

函数 newrb 利用迭代方法设计径向基函数网络，该方法每迭代一次就增加一个神经元，直到平方和误差下降到目标误差以下或隐层神经元个数达到最大值时迭代停止。函数 newrb 的调用形式为

net = newrb (P , T , GOAL , SPREAD, MN , DF)

其中，GOAL 表示目标误差，MN 表示最大神经元个数，DF 表示迭代过程的显示频率。

2.5.2 广义回归网络

1. 广义回归网络的结构

广义回归网络（GRNN）的结构与径向基函数网络类似，如图 2.24 所示。该网络的隐层和输出层的神经元个数均与输入样本矢量的个数相同，其输出层是特殊线性层。图 2.24 中的 nprod 模块表示计算隐层输出矢量 a^1 和输出层权值矩阵 W^2 的规则化内积，



即对内积所得矢量中的每一元素分别除以 a^1 中各元素的总和，因此广义回归网络是一种规则化的径向基函数网络。

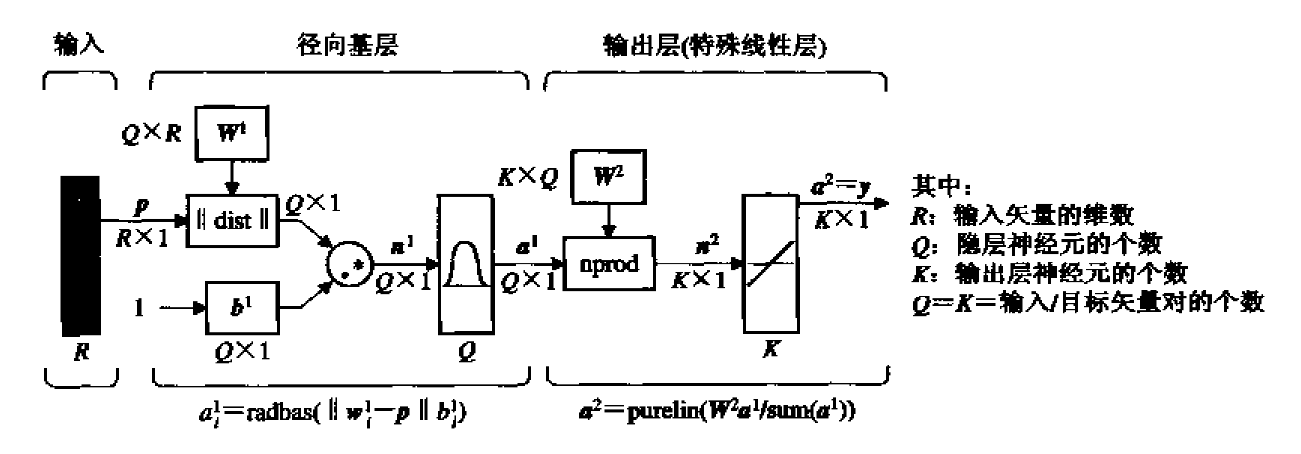


图 2.24　广义回归网络结构图

广义回归网络常用于解决函数逼近问题，当隐层神经元足够多时，该网络能够以任意精度逼近一个平滑函数，其缺点在于当输入样本数目很多时，网络十分庞大，计算复杂，因此不适用于训练样本数目过多的情况。

2. 广义回归网络的设计

广义回归网络可利用函数 newgrnn 进行设计，其调用形式为

net = newgrnn (P , T , SPREAD)

其中，P、T 分别为输入样本矢量集和目标输出矢量集构成的矩阵；SPREAD 是扩展常数，其缺省值为 1；net 为函数生成的广义回归网络对象。

2.5.3　概率神经网络

1. 概率神经网络的结构

概率神经网络是径向基网络的另一种重要变形，其结构如图 2.25 所示。该网络的隐层神经元个数与输入样本矢量的个数相同，输出层神经元个数等于训练样本数据的种类个

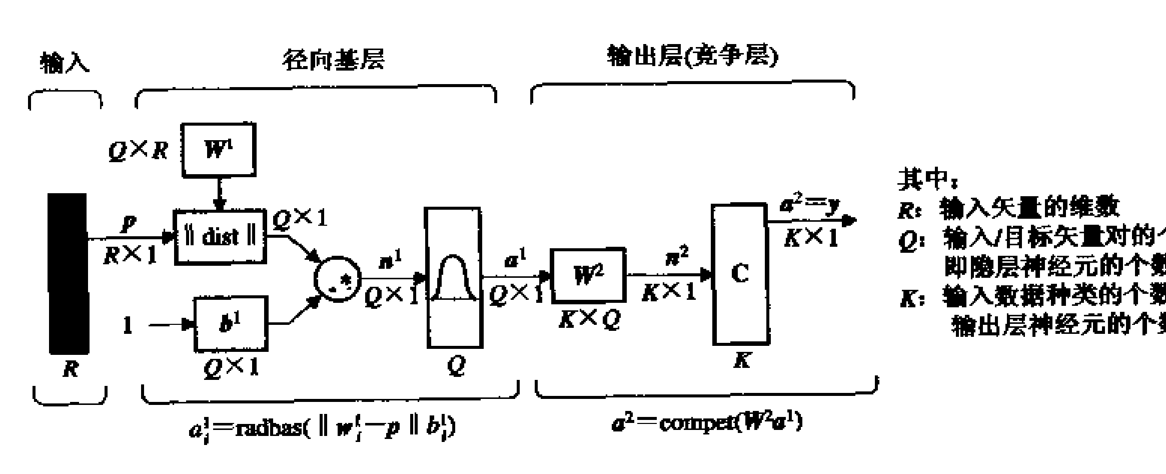
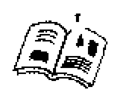


图 2.25　概率神经网络结构图



数。该网络的输出层是竞争层，每个神经元分别对应于一个数据类别。图中的模块 C 表示竞争传递函数，其功能是找出其输入矢量 n^2 中各元素的最大值，并且使与最大值对应类别的神经元输出为 1，其他类别的神经元输出为 0。这种网络得到的分类结果能够达到最大的正确概率。

概率神经网络常用来解决分类问题。当训练样本数据足够多时，概率神经网络收敛于一个贝叶斯分类器，而且推广性良好。其缺点与广义回归网络相同，即当输入样本数目过多时，计算将变得复杂，因此运算速度比较缓慢。

2. 概率神经网络的设计

概率神经网络的设计函数为 newpnn ，其调用形式为

```
net = newpnn ( P , T , SPREAD )
```

其中，目标矢量矩阵 T 的每行元素中只含有一个 1 ，其余均为 0 ，1 的位置标号表示与该行对应的样本矢量的类别。扩展常数 SPREAD 的缺省值为 0.1 ，当 SPREAD 趋近于 0 时，网络趋近于近邻分类器；当 SPREAD 趋近于 ∞ 时，网络趋近于线性分类器。

2.6 自组织网络

在生物神经细胞中存在一种特征敏感细胞，这种细胞只对外界信号刺激的某一特征敏感，并且这种特性是通过自学习形成的。在人脑的脑皮层中，对外界信号刺激的感知和处理是分区进行的。有学者认为，脑皮层是通过邻近神经细胞的相互竞争学习，自适应地发展成为对不同性质信号敏感的区域。根据这些现象，芬兰学者 Kohonen 提出了自组织特征映射网络模型（Self-Organizing Feature Map）。他认为，一个神经网络在接受外界输入模式时，会自适应的对输入信号特征进行学习，进而自组织地形成不同区域，并且各区域对输入模式将具有不同的响应特征。在输出空间中，这些神经元将形成一张映射图，映射图中功能相同的神经元靠得较近，功能不同的神经元分得较开，自组织特征映射网络也由此而得名。

自组织特征映射过程是通过竞争学习完成的。竞争学习是指同一层神经元之间互相竞争，竞争胜利的神经元修改与其相联的连接权值的过程。竞争学习是一种无监督学习方法，在学习过程中，只需向网络提供一些学习样本，而无需提供理想的目标输出。网络根据输入样本的特性进行自组织映射，从而对样本进行自动排序和分类。

自组织网络在样本排序、样本检测和样本分类方面有着广泛的应用。MATLAB 6.x 神经网络工具箱中提供了竞争学习网络、自组织映射网络以及学习矢量量化网络，它们都主要应用于分类方面，可分别用函数 newc 、newsom 和 newlvq 进行设计。



2.6.1　竞争学习网络

1. 竞争学习网络的结构

竞争学习网络的结构如图 2.26 所示，该网络具有 R 维输入和 S 个输出，由隐层和竞争层组成。图中的‖ndist‖模块表示对输入矢量 $\boldsymbol{p}$ 和神经元权值矢量 $\boldsymbol{w}$ 之间的距离取负。该网络的输出层是竞争层，图中的模块 C 表示竞争传递函数，其输出矢量由竞争层各神经元的输出组成，除在竞争中获胜的神经元外，其余神经元的输出都为零。竞争传递函数输入矢量 $\boldsymbol{n}$ 中的最大元素所对应的神经元是竞争中的获胜者，其输出固定为 1。

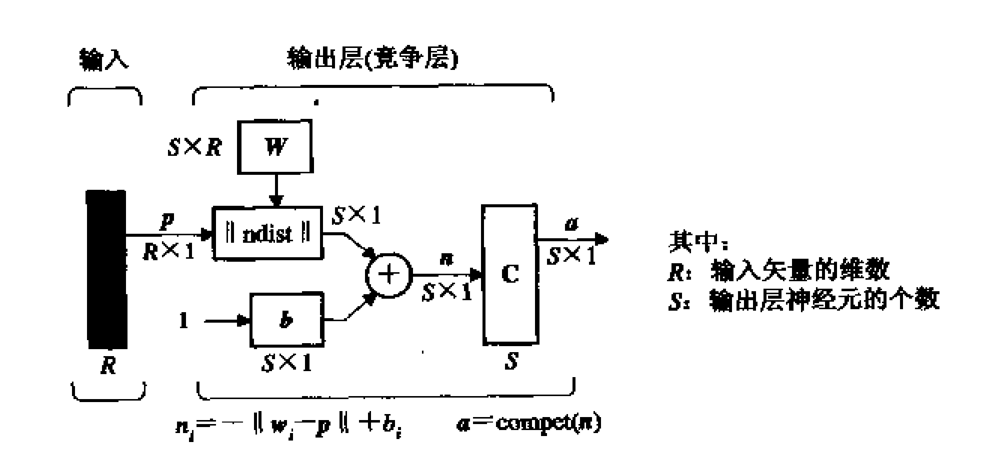


图 2.26　竞争学习网络结构图

2. 竞争学习网络的建立

函数 newc 可用来建立一个竞争学习网络。函数调用形式为

net = newc (PR , S , KLR , CLR)

其中，PR 为 R 维输入矢量中每个元素可取的最小值和最大值所构成的 $R\times2$ 维矩阵；S 为神经元个数，也是类别个数；KLR 表示 Kohonen 学习速率，其缺省值为 0.01；CLR 为阈值学习速率，其缺省值为 0.001；函数 newc 生成的竞争学习网络经过训练后才能使用。

3. 竞争学习网络的训练

竞争学习网络依据 Kohonen 学习规则和阈值学习规则进行训练。

竞争学习网络每进行一步学习，权值矢量与当前输入矢量最为接近的神经元将在竞争中取胜，网络则依据 Kohonen 学习规则对这个神经元的权值进行调整。假定竞争层中的第 i 个神经元获胜，其权值矢量 $\boldsymbol{w}_i$ 将修改为

$$\boldsymbol{w}_i(q)=\boldsymbol{w}_i(q-1)+\alpha(\boldsymbol{p}(q)-\boldsymbol{w}_i(q-1))$$

按照上述规则，修改后的获胜神经元权值矢量将更接近于当前的输入。经过这样的调整后，当下一次网络输入类似的矢量时，这一神经元就极有可能在竞争中取胜；若输入的矢量与该神经元的权值矢量相差很大时，则该神经元极有可能落败。随着训练的进行，网



络中的每一个神经元将代表一类近似的矢量，当接受某类矢量输入时，对应类别的神经元将在竞争中获胜，从而网络就具有了分类的功能。Kohonen 学习规则对应的工具箱函数为 learnk。

如果竞争学习网络中某一神经元的初始权值被设置得远离所有训练样本矢量，那么这一神经元在训练过程中将没有机会取胜，其权值将一直得不到调整，在训练结束后依然不能表示某类数据特性，这样的神经元在网络中是无用的。为避免这种情况的发生，在训练中可以有意地改变各神经元的阈值，使得经常取胜的神经元的阈值变小，不常取胜的神经元的阈值变大，从而使各神经元获胜的机会均等。工具箱函数 learncon 可用来执行这种阈值学习规则。需要注意的是，阈值学习速率的设置应小于 Kohonen 学习速率。

在建立竞争学习网络时，只要训练函数设置正确，网络在训练过程中将自动采用上述学习规则。假设神经网络对象的名称为 net，只要 net 的训练函数设为 trainr 函数，即

```
net.trainFcn = 'trainr'
```

那么在调用训练函数 train 时，网络将自动按照上述规则进行训练，即

```
net = train ( net , P )
```

其中，P 为网络的输入矢量矩阵。

2.6.2 自组织特征映射网络

自组织特征映射网络（SOFM）的构造是基于对人类脑皮层功能的模仿。在人脑的脑皮层中，对外界信号刺激的感知和处理是分区进行的，因此自组织特征映射网络不仅要对不同输入信号产生不同的响应，即与竞争学习网络一样具有分类的功能，更重要的是要实现功能相同的神经元在空间分布上的聚集。因此自组织映射网络在训练时除了要对获胜神经元的权值进行调整外，还要对获胜神经元邻域内的所有神经元进行权值修正，从而使邻近的神经元具有相同的功能。

1. 自组织特征映射网络的结构

自组织特征映射网络的结构如图 2.27 所示，其结构与竞争学习网络完全相同，但自组织映射网络在训练时要对获胜神经元邻域内的所有神经元进行权值的修正。

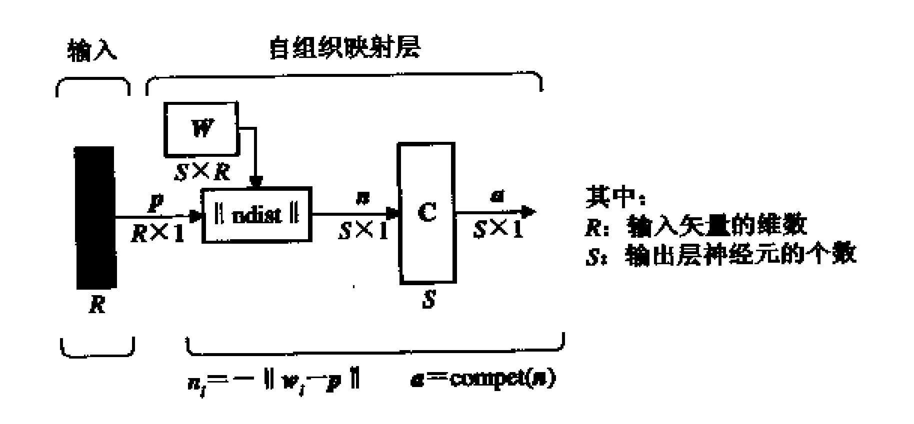


图 2.27 自组织特征映射网络结构图



2. 输出层神经元的拓扑结构和距离计算

自组织特征映射网络输出层的神经元可以按任意维形式排列，工具箱函数 gridtop、hextop 和 randtop 可用来建立输出层神经元在物理位置上的拓扑结构，它们分别把神经元设置在长方形、六角形或任意形状的网格上，这三个函数的调用形式分别为

pos = gridtop (dim1, dim2, … , dimN)

pos = hextop (dim1, dim2, … , dimN)

pos = randtop (dim1, dim2 , … , dimN)

其中，dim1，dim2，…，dimN 是以 N 维形式排列的输出层神经元在各维的个数；函数返回值 pos 是一个 $N \times S$ 维矩阵，其中 S 等于 dim1、dim2、… 、dimN 的乘积，是输出层神经元的总个数，矩阵 pos 中的每一列表示一个神经元在 N 维空间中的坐标。函数 plotsom 可以用来描绘输出层神经元的拓扑图，其调用形式如下：

plotsom (pos)

图 2.28 给出了这三种拓扑结构的示意图。

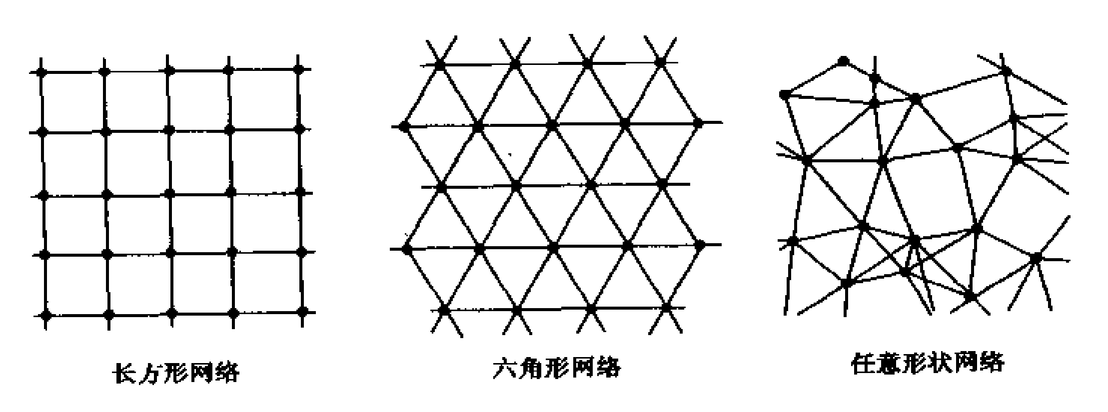


图 2.28　输出层神经元拓扑结构示意图

工具箱函数 dist、boxdist、linkdist 或 mandist 可用来计算自组织特征映射网络输出层神经元之间的距离，这些函数的调用形式如下：

D = dist (pos)

D = boxdist (pos)

D = linkdist (pos)

D = mandist (pos)

其中，pos 表示神经元位置的 $N \times S$ 维矩阵，矩阵的每一列对应一个神经元的位置坐标；返回值 D 为 $S \times S$ 维矩阵，矩阵中的每一个元素 D(i, j)表示第 i 个和第 j 个神经元之间的距离。这四种距离函数采用的计算方法如表 2.4 所示。表中利用 MATLAB 语句表示各距离函数的计算方法，并且假定第 i 个和第 j 个神经元的位置坐标矢量分别为 P_i 和 P_j，两神经元之间的距离为 D_{ij} 。

在设定了网络拓朴结构和距离函数之后，可以确定神经元的邻域。图 2.29 为某一神经元的邻域示意图，图中的输出层神经元拓扑结构由 gridtop 函数生成，假定位于中心的神经元为竞争获胜神经元，当采用不同的距离函数时，该神经元的邻域形状不同，其中左图采用的距离函数为 boxdist ，右图采用的距离函数为 linkdist。



表 2.4　距离函数及计算方法

距离函数	距离计算方法
dist	Dij = sum ((Pi - Pj).^ 2) . ^ 0.5
boxdist	Dij = max (abs (Pi – Pj))
linkdist	Dij = 0　如果 i==j Dij = 1　如果 ((sum ((Pi – Pj) . ^ 2)) . ^ 0.5) ≤ 1 Dij = 2　如果存在 k，使 Dik = Dkj = 1 成立 Dij = 3　如果存在 k_1、k_2，使 $Dik_1 = Dk_1k_2 = Dk_2j = 1$ 成立 Dij = M　如果存在 k_1、k_2、…、k_M，使 $Dik_1 = Dk_1k_2 = \cdots = Dk_Mj = 1$ 成立 Dij = S　其他
mandist	Dij = sum (abs (Pi – Pj))

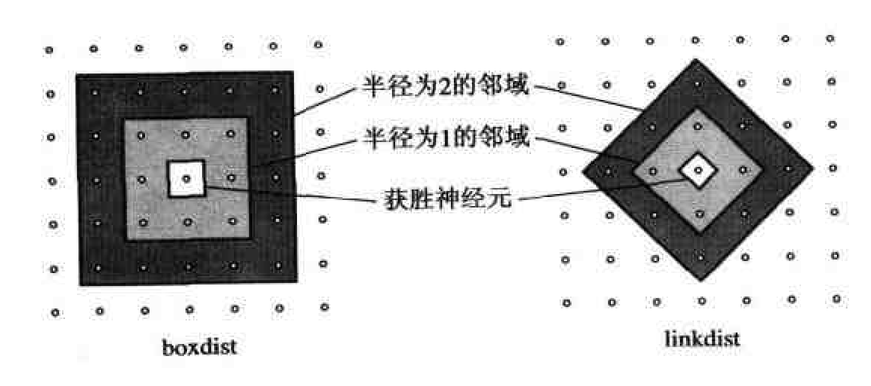


图 2.29　神经元邻域示意图

3. 自组织特征映射网络的建立

函数 newsom 可用来建立一个自组织特征映射网络，其调用形式为

```
net = newsom ( PR , D , TFCN , DFCN , OLR , OSTEPS , TLR , TND )
```

其中，PR 为 R 维输入矢量中每维输入可取的最小值和最大值所构成的 $R\times2$ 维矩阵；D 为输出层神经元在多维空间中排列时各维的个数，缺省值为[5 8]；TFCN 用于设定拓扑函数，缺省值为'hextop'；DFCN 用于设定距离函数，缺省值为'linkdist'；OLR 用于排列阶段学习速率，缺省值为 0.9；OSTEPS 用于排列阶段学习次数，缺省值为 1000；TLR 用于调整阶段学习速率，缺省值为 0.02；TND 用于调整阶段邻域半径，缺省值为 1。

4. 自组织特征映射网络的训练

自组织特征映射网络的训练过程分排列和调整两个阶段进行。在网络的训练过程中，假定当前输入下第 $i*$ 个神经元获胜，那么网络中神经元权值的修正将依照下式进行

$$w_{i^*}(q) = w_{i^*}(q-1) + \alpha(p(q) - w_{i^*}(q-1))$$

$$w_j(q) = w_j(q-1) + 0.5\alpha(p(q) - w_j(q-1)) \qquad j \in N_{i^*}(d)$$

其中，$N_{i^*}(d)$ 表示获胜神经元的邻域，即

$$N_{i^*}(d) = \{j, d_{i^*j} \le d\}$$



从上述公式中可以看出，获胜神经元权值的修正量与学习速率成正比，邻域内神经元权值的修正量与学习速率的一半成正比，而邻域外的神经元权值不作修正。在排列和调整这两个阶段的训练中，学习速率和邻域半径的设定有所不同。

在排列阶段中，邻域半径首先设定为两个神经元的最大可能距离，然后在训练过程中逐渐减小到指定的调整阶段邻域半径；而学习速率首先采用排列阶段学习速率，然后在训练过程中逐步降低到调整阶段学习速率。在这一阶段中，网络中的神经元按照输入数据的分布进行排列，从而实现了功能相同的神经元在输入空间分布上的聚集。排列阶段的学习次数在网络建立时由设计者指定。

在调整阶段中，邻域半径一直保持为指定的调整阶段邻域半径，一般取值较小，典型值为 1.0，学习速率则从调整阶段学习速率开始缓慢降低。小邻域半径和缓慢下降的学习速率可以对网络进行良好的调整，同时又不破坏排列阶段神经元已形成的分布。由于调整过程非常缓慢，因此调整阶段的学习次数要远大于排列阶段的学习次数。

上述学习规则可由工具箱函数 learnsom 实现。当网络的训练函数设置为'trainr'或自适应函数设置为'trains'时，即

```
net.trainFcn = 'trainr'  或  net.adaptFcn = 'trains'
```

那么在调用训练函数 train 时，网络将按上述学习规则进行训练，即

```
net = train ( net , P )
```

其中，P 为网络的输入矢量矩阵。

2.6.3　学习矢量量化网络

1. 学习矢量量化网络的结构

学习矢量量化网络由一个竞争层和一个线性层组成，结构如图 2.30 所示。在该网络中，竞争层的作用仍是分类，但它首先将输入矢量划分为较精细的子类别，然后线性层将竞争层的分类结果进行合并，从而形成符合用户定义的目标分类模式，因此线性层的神经元个数 S^2 要比竞争层的神经元个数 S^1 少。

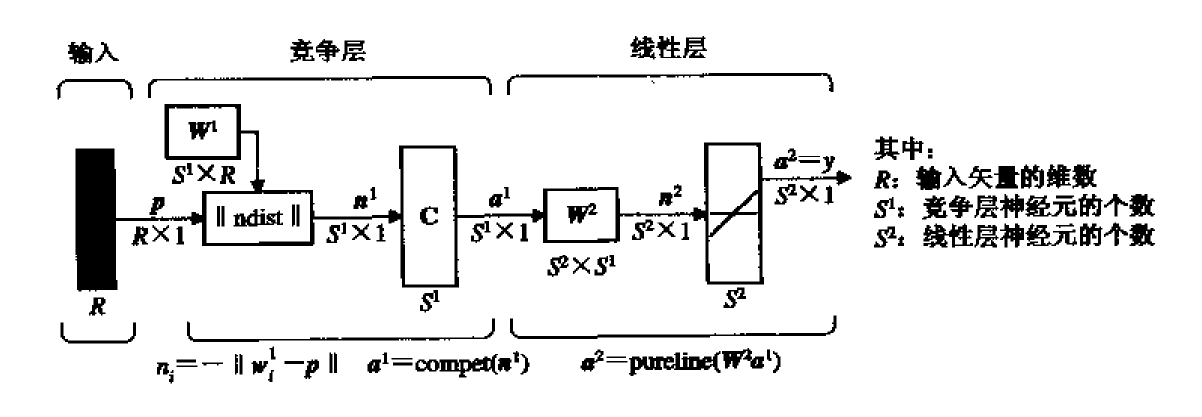


图 2.30　学习矢量量化网络结构图



2. 学习矢量量化网络的建立

学习矢量量化网络可以由函数 newlvq 建立，该函数的调用形式为

```
net = newlvq ( PR , S1 , PC , LR , LF )
```

其中，PR 为 R 维输入矢量中每维输入可取的最小值和最大值所构成的 $R\times2$ 维矩阵；S1 为竞争层神经元的个数；PC 是一个 S^2 维矢量，S^2 为数据类别个数，PC 中的每一个元素分别表示对应类别的输入样木矢量个数占输入样本矢量总数的百分比；LR 为学习速率，缺省值为 0.01；LF 为学习函数，可以是'learnlv1'或'learnlv2'，缺省值为'learnlv1'。

3. 学习矢量量化网络的训练

学习矢量量化网络在建立时，竞争层和线性层之间的连接权值矩阵 W^2 就已经确定了，如果与竞争层某一神经元对应的矢量子类别属于线性层某个神经元所对应的目标类别，则这两个神经元的连接权值为 1 ，否则二者的连接权值为 0 ，这样的权值矩阵就实现了子类别到目标类别的合并。根据上述办法，W^2 的每一列中除了一个元素为 1 之外，其余元素均为 0，1 在该列中的位置表明了竞争层所确定的子类别属于哪种目标类别。在建立网络时，每类数据占数据总数的百分比是已知的，这也是竞争层神经元归并到线性层各个输出时所依据的比例。由于 W^2 是事先确定的，所以在网络训练时只需要调整竞争层的权值矩阵 W^1 ，同时，训练结果不仅要保证竞争层实现正确的分类，还要保证竞争层神经元的排列顺序与线性层神经元的权值矩阵正确对应。

在网络的训练过程中，假定当前输入下的第 i*个神经元获胜，那么网络中神经元权值的修正将依照下式进行：

$$\boldsymbol{w}_{i^*}^1(q)=\boldsymbol{w}_{i^*}^1(q-1)+\alpha(\boldsymbol{p}(q)-\boldsymbol{w}_{i^*}^1(q-1)) \qquad \text{若分类正确}$$

$$\boldsymbol{w}_{i^*}^1(q)=\boldsymbol{w}_{i^*}^1(q-1)-\alpha(\boldsymbol{p}(q)-\boldsymbol{w}_{i^*}^1(q-1)) \qquad \text{若分类错误}$$

上式表明，当分类正确时，竞争层划分的子类别被正确归并到目标类别中，于是获胜神经元权值将被调整得更加逼近输入矢量；当分类错误时，竞争层划分的子类别未被正确归并，即获胜神经元的位置与线性层神经元权值矩阵没有正确对应，于是获胜神经元权值在调整后将远离输入矢量。上述的学习算法被称作 LVQ1 学习规则，工具箱函数 learnlv1 实现的正是这一规则。

在应用 LVQ1 学习规则对网络进行训练后，假定某一输入矢量落入到两个竞争层神经元权值矢量的区域中间，而且这两个权值矢量中的一个将带来正确分类，另一个将会导致错误分类，于是这个输入矢量被错分的可能性会较大。为避免这种情况的发生，可以采用 LVQ2 学习规则进一步对网络进行训练。在 LVQ2 学习规则中，假定当前输入矢量位于竞争层神经元 i* 和 j* 的权值矢量的中间区域，即

$$\min\left(\frac{d_{j^*}}{d_{i^*}},\frac{d_{i^*}}{d_{j^*}}\right)>s$$

其中

$$s=\frac{1-w}{1+w}$$



上式中的 d_{i^*} 和 d_{j^*} 分别表示输入矢量到两个权值矢量间的欧式距离，w 的值通常取在 0.2 和 0.3 之间，同时神经元 i^* 获胜将导致错分，神经元 j^* 获胜将带来正确分类。在上述条件同时满足的情况下，神经元权值将作如下修正：

$$\boldsymbol{w}_{i^*}^{1}(q)=\boldsymbol{w}_{i^*}^{1}(q-1)-\alpha(\boldsymbol{p}(q)-\boldsymbol{w}_{i^*}^{1}(q-1))$$

$$\boldsymbol{w}_{j^*}^{1}(q)=\boldsymbol{w}_{j^*}^{1}(q-1)+\alpha(\boldsymbol{p}(q)-\boldsymbol{w}_{j^*}^{1}(q-1))$$

上述学习规则可以使网络的稳定性增强，从而减少了输入矢量被错分的可能。工具箱函数 learnlv2 用于实现这一学习规则。

在建立学习矢量量化网络时，如果指定的学习函数为 learnlv2，那么在网络训练时将先使用函数 learnlv1 对网络进行训练，然后再使用基于 LVQ2 学习规则的 Learnlv2 函数对网络进行进一步的训练。

2.7 反馈网络

前面几节介绍的网络都属于前向网络，这一节将介绍神经网络的另一个重要类型——反馈网络。在反馈网络中，信息在前向传递的同时还要进行反向传递，这种信息的反馈可以发生在不同网络层神经元之间，也可以只限于某一层神经元上。由于反馈网络是动态网络，因此只有满足了稳定性条件，网络才能在工作一段时间后达到稳定状态。

MATLAB 6.x 神经网络工具箱中提供的反馈网络包括 Elman 网络和 Hopfield 网络，它们可分别用函数 newelm 和 newhop 建立，其中 Elman 网络主要用于信号检测和预测等方面，Hopfield 网络主要用于联想记忆、聚类和优化计算等方面。

2.7.1 Elman 网络

1. Elman 网络的结构

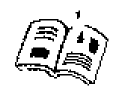
Elman 网络由若干个隐层和输出层构成，并且在隐层存在反馈环节，其结构如图 2.31 所示。图中，模块 **D** 表示时延环节，隐层神经元采用 tansig 函数作为传递函数，输出层传递函数为纯线性函数 purelin，这两层神经元的传递函数可以在建立网络时由用户自己指定。当隐层神经元足够多时，Elman 网络的这种结构可以保证网络以任意精度逼近任意的非线性函数。

2. Elman 网络的建立

Elman 网络可以由工具箱函数 newelm 建立，该函数的调用形式为

net = newelm (PR , [S1 S2 ··· SN] , { TF1 TF2 ··· TFN }, BTF , BLF , PF)

其中，PR 为 R 维输入矢量中每维输入可取的最小值和最大值所构成的 $R\times 2$ 维矩阵；S1 到 SN 分别表示各层神经元的个数；TF1 到 TFN 为各层神经元的传递函数；BTF 为网络的训练函数，缺省值为'traingdx'；BLF 是学习规则函数，缺省值为'learngdm'；PF 是网络性能



指标函数，缺省值为'mse'。

Elman 网络的初始化函数在网络生成时设定为 initnw，即 Nguyen-Widrow 初始化规则。网络在训练时采用基于误差反向传播算法的学习函数，如 trainlm、trainbfg、trainrp、traingd 等。

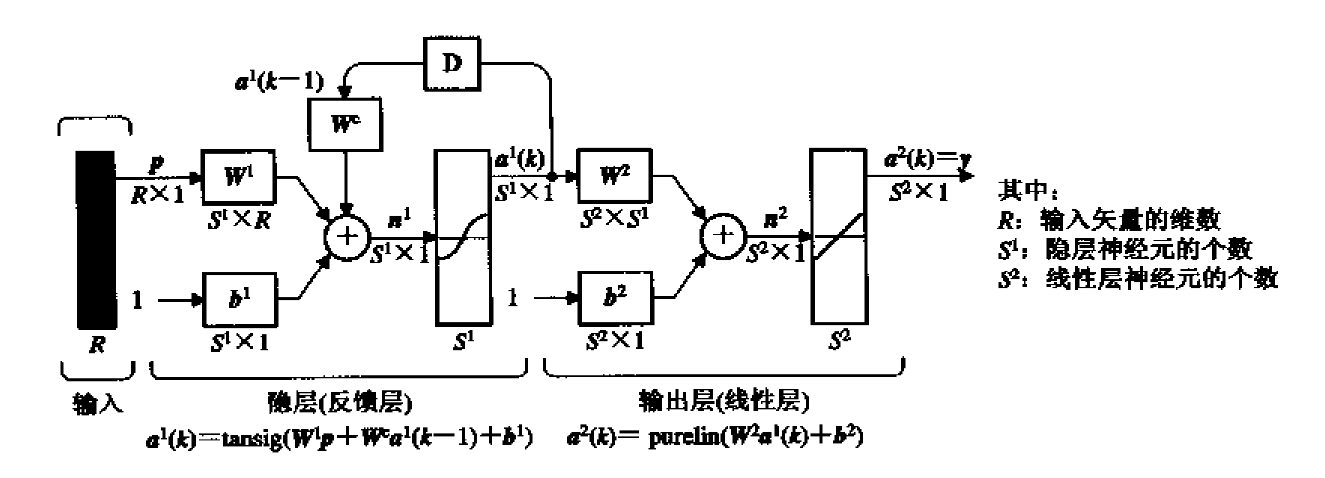


图 2.31 Elman 网络结构图

2.7.2 Hopfield 网络

Hopfield 网络主要用于联想记忆和优化计算等问题。联想记忆是指当网络输入某个矢量后，网络经过反馈演化，从网络输出端得到另一个矢量，这样输出矢量称作网络从初始输入矢量联想得到的一个稳定记忆，即网络的一个平衡点。优化计算是指当某一问题存在多种解法时，可以设计一个目标函数，然后寻求满足这一目标函数的最优解法。例如，在很多情况下可以把能量函数作为目标函数，得到的最优解法需要使能量函数达到极小点，即能量函数的稳定平衡点。总之，Hopfield 网络的设计思想就是在初始输入下，使网络经过反馈计算最后达到稳定状态，这时的输出即是用户需要的平衡点。

1. Hopfield 网络的结构

单层 Hopfield 网络的结构如图 2.32 所示，其神经元传递函数为对称饱和线性函数

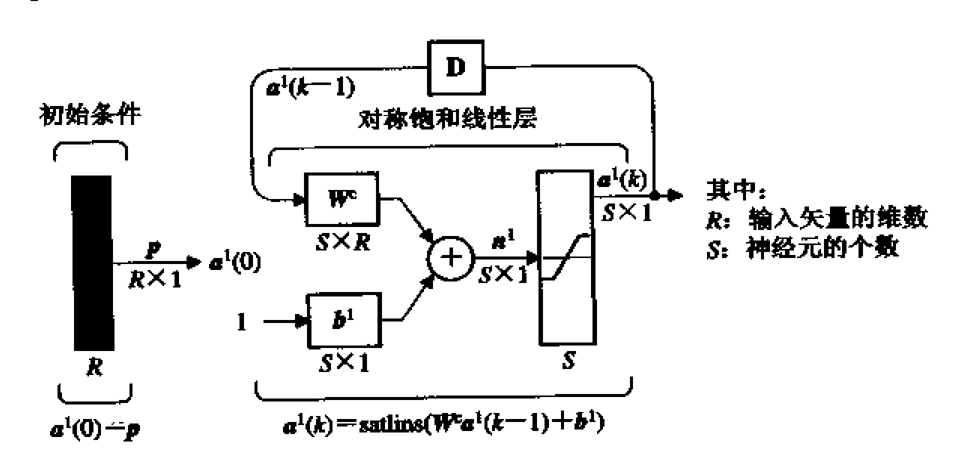


图 2.32 Hopfield 网络结构图



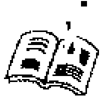
satlins。网络的输入作为初始值首先输入到各神经元，经过反馈计算，网络最终稳定在某一状态，并输出这一稳定值。

2. Hopfield 网络的设计

newhop 函数可用来设计 Hopfield 网络，该函数的调用方式为

net = newhop (T)

其中，T 为日标矢量，也就是用户希望网络所能达到的平衡点。需要注意的是，newhop 函数可能会导致虚假平衡点的出现。







第三章 基于 MATLAB 6.x 的神经网络设计与分析

在 MATLAB 神经网络工具箱中，神经网络对象是一个非常重要的概念。工具箱的设计者们在神经网络对象中封装了网络结构、网络权值和阈值以及训练函数等所有重要的网络属性，当用户建立一个神经网络对象后，只要设置好网络属性，就可以方便地使网络按照自己的期望进行训练和工作，而不必再编写冗长的程序语句，从而大大提高了神经网络系统的设计与分析效率。

在 MATLAB 中，用户不仅可以利用工具箱函数进行编程来实现神经网络的设计和分析，还可以利用图形用户界面 GUI 和动态可视化仿真工具 Simulink 方便直观地对神经网络进行设计和分析，这两种方法的本质虽然仍是基于神经网络对象和工具箱函数，但却省略了编程过程。本章将首先介绍神经网络对象的结构和属性，然后对神经网络工具箱函数进行说明，最后介绍如何利用图形用户界面 GUI 和 Simulink 工具对神经网络进行设计和分析。

3.1 神经网络对象

一个神经网络对象定义和存储了网络总体结构、子对象结构（网络细节结构）、函数、函数参数、权值和阈值、其他数据共六部分的网络属性参数，本节将介绍这些属性参数的具体含义。下文中均假定神经网络对象的名称为 net。

3.1.1 神经网络对象的属性

1. 网络总体结构

网络总体结构包括以下属性：

(1) numInputs：该属性定义了神经网络的输入个数，属性值可以是 0 或正整数。需要注意的是，该属性定义了网络输入矢量的总个数，而不是单个输入矢量的维数。

(2) numLayers：该属性定义了神经网络的层数，其属性值可以是 0 或正整数。

(3) biasConnect：该属性定义了神经网络的每层是否具有阈值，其属性值为 $N\times1$ 维的布尔量矩阵，其中 N 为网络的层数(net.numLayers)。net.biasConnect(i)为 1，表示第 i 层上的神经元具有阈值，为 0 则表示该层没有阈值。



(4) inputConnect：该属性定义了神经网络的输入层，其属性值为 $N \times Ni$ 维的布尔量矩阵，其中 N 为网络的层数，Ni 为网络的输入个数(net.numInputs)。net.inputConnect(i,j)为 1，表示第 i 层上的每个神经元都要接收网络的第 j 个输入矢量，为 0 则表示不接收该输入。

(5) layerConnect：该属性定义了网络各层的连接情况，其属性值为 $N \times N$ 维的布尔量矩阵，其中 N 为网络的层数。net.layerConnect(i,j)为 1，表示第 i 层与第 j 层上的神经元相连，为 0 则表示它们不相连。

(6) outputConnect：该属性定义了神经网络的输出层，其属性值为 $1 \times N$ 维的布尔量矩阵，其中 N 为网络的层数。net.outputConnect(i)为 1，表示第 i 层神经元将产生网络的输出，为 0 则表示该层不产生输出。

(7) targetConnect：该属性定义了神经网络的目标层，即网络哪些层的输出具有目标矢量。其属性值为 $1 \times N$ 维的布尔量矩阵，其中 N 为网络的层数。net.targetConnect(i)为 1，表示第 i 层神经元产生的输出具有目标矢量，为 0 则表示该层输出不具有目标矢量。

(8) numOutputs：该属性定义了神经网络输出矢量的个数，属性值为只读变量，其数值为网络中输出层的总数(sum(net.outputConnect))。

(9) numTargets：该属性定义了网络目标矢量的个数，属性值为只读变量，其数值为网络中目标层的总数(sum(net.targetConnect))。

(10) numInputDelays：该属性定义了神经网络的输入延迟，属性值为只读变量，其数值为网络各输入层输入延迟拍数(net.inputWeights{i,j}.delays)中的最大值。

(11) numLayerDelays：该属性定义了神经网络的层输出延迟，属性值为只读变量，其数值为各层的神经元之间连接延迟拍数(net.layerWeights{i,j}.delays)中的最大值。

2. 子对象结构

子对象结构包括以下属性：

(1) inputs：该属性定义了神经网络每个输入的属性，其属性值为 $Ni \times 1$ 维的单元数组，其中 Ni 为网络输入的个数。

(2) layers：该属性定义了神经网络每层神经元的属性，其属性值为 $N \times 1$ 维的单元数组，其中 N 为网络的层数。

(3) outputs：该属性定义了神经网络每个输出的属性，其属性值为 $N \times 1$ 维的单元数组，其中 N 为网络的层数。

(4) targets：该属性定义了每层神经网络目标矢量的属性，其属性值为 $N \times 1$ 维的单元数组，其中 N 为网络的层数。

(5) biases：该属性定义了每层神经网络阈值的属性，其属性值为 $N \times 1$ 维的单元数组，其中 N 为网络的层数。

(6) inputWeights：该属性定义了神经网络每组输入权值的属性，其属性值为 $N \times Ni$ 维的单元数组，其中 N 为网络的层数，Ni 为网络的输入个数。

(7) layerWeights：该属性包括了神经网络各层神经元之间网络权值的属性，其属性值为 $N \times N$ 维的单元数组，其中 N 为网络的层数。



3. 函数

神经网络对象的函数包括以下属性：

(1) adaptFcn：该属性定义了网络的自适应调整函数，其属性值为表示自适应函数名称的字符串。

(2) initFcn：该属性定义了网络的初始化函数，其属性值为表示网络初始化函数名称的字符串。

(3) performFcn：该属性定义了衡量网络输出误差的性能函数，其属性值为表示性能函数名称的字符串。

(4) trainFcn：该属性定义了网络的训练函数，其属性值为表示训练函数名称的字符串。

4. 函数参数

神经网络对象的函数参数包括以下属性：

(1) adaptParam：该属性定义了网络当前自适应函数的各参数，其属性值为各参数构成的结构体。

(2) initParam：该属性定义了网络当前初始化函数的各参数，其属性值为各参数构成的结构体。

(3) performParam：该属性定义了网络当前性能函数的各参数，其属性值为各参数构成的结构体。

(4) trainParam：该属性定义了网络当前训练函数的各参数，其属性值为各参数构成的结构体。

5. 权值和阈值

神经网络对象的权值和阈值包括以下属性：

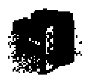
(1) IW：该属性定义了网络输入和各输入层神经元之间的网络权值，属性值为 $N \times Ni$ 维的单元数组，其中，N 为网络的层数，Ni 为网络的输入个数。如果 net.inputConnect(i,j) 为 1，即第 i 层上的各神经元接收网络的第 j 个输入，那么在单元 net.IW{i,j}中将存储它们之间的网络权值矩阵，该矩阵的行数为第 i 层神经元的个数(net.layers{i}.size)，列数为第 j 个输入的维数(net.inputs{j}.size)与j输入延迟拍数(net.inputWeights{i,j}.delays)的乘积。

(2) LW：该属性定义了各层神经元之间的网络权值，其属性值为 $N \times N$ 维的单元数组，其中 N 为网络的层数。如果 net.layerConnect(i,j)为 1，即第 i 层与第 j 层上的神经元相连，那么在单元 net.LW{i,j}中将存储它们之间的网络权值矩阵，该矩阵的行数为第 i 层上神经元的个数(net.layers{i}.size)，列数为第 j 层神经元的个数(net.layers{j}.size)与层输入延迟拍数(net.layerWeights{i,j}.delays)的乘积。

(3) b：该属性定义了各层神经元的阈值，其属性值为 $1 \times N$ 维的单元数组，其中 N 为网络的层数。如果 net.biasConnect(i)为 1，即第 i 层的神经元具有阈值，那么单元 net.b{i}中将存储该层的阈值矢量，矢量维数和第 i 层的神经元个数(net.layers{j}.size)相等。

6. 其他数据

userdata 属性用于存储用户信息。



3.1.2　神经网络的细节结构（子对象属性）

本节将详细介绍网络输入、网络层、输出、目标、权值和阈值等子对象的属性。

1. 输入(inputs)

网络输入 net.inputs 为 $Ni \times 1$ 维的单元数组，其中 net.inputs{i}中存储了网络第 i 个输入的信息。网络的每个输入具有以下属性：

(1) size：该属性定义了输入矢量的维数，其属性值可以为0或正整数。

(2) range：该属性定义了输入矢量的取值范围，其属性值为 $Ri \times 2$ 维的矩阵，Ri 为输入矢量的维数(net.inputs{i}.size)，该矩阵中的每一行表示输入矢量中各维元素可取的最小值和最大值。

(3) userdata：该属性用于存储用户信息。

2. 层(layers)

网络各层 net.layers 为 $N \times 1$ 维的单元数组，其中 net.layers{i}中存储了网络第 i 层的信息。网络的每一层具有以下属性：

(1) size：该属性定义了每层神经元的个数，其属性值可以为0或正整数。

(2) dimensions：该属性定义了每层神经元在多维空间中排列时各维的维数，其属性值为一个行矢量，该矢量中各元素的乘积等于该层神经元的个数(net.layers{i}.size)。该属性对于自组织映射网络十分重要。

(3) topologyFcn：该属性定义了每层神经元在多维空间中分布的拓扑函数，其属性值为表示拓扑函数名称的字符串。

(4) positions：该属性定义了每层网络中各神经元的位置坐标，属性值为只读变量，其值由拓扑函数(net.layers{i}.topologyFcn)和神经元在各维分布的维数(net.layers{i}.dimensions)来确定。

(5) distanceFcn：该属性定义了每层神经元间距的计算函数，其属性值为表示距离函数名称的字符串。

(6) distances：该属性定义了每层网络中各神经元之间的距离，属性值为只读变量，其数值由神经元的位置坐标(net.layers{i}.positions)和距离函数(net.layers{i}.distanceFcn)来确定。

(7) initFcn：该属性定义了每层神经元的初始化函数，其属性值为表示初始化函数名称的字符串。

(8) netInputFcn：该属性定义了每层神经元的输入函数，其属性值为表示输入函数名称的字符串。该函数确定了网络每层的加权输入以何种方式和阈值组合在一起以形成神经元传递函数的输入。

(9) transferFcn：该属性定义了每层神经元的传递函数，其属性值为表示传递函数名称的字符串。

(10) userdata：该属性用于存储用户信息。



3. 输出(outputs)

网络输出 net.outputs 为 $N\times1$ 维的单元数组，如果第 i 层产生网络的一个输出，那么在 net.outputs{i}中将存储该输出量的信息。网络的每一个输出都具有以下属性：

(1) size：该属性定义了每一个网络输出的维数，属性值为只读变量，其数值等于该输出层中神经元的个数(net.layers{i}.size)。

(2) userdata：该属性用于存储用户信息。

4. 目标(targets)

目标 net.targets 为 $N\times1$ 维的单元数组，如果第 i 层具有网络的一个目标矢量，那么在 net.targets{i}中将存储该目标矢量的信息。网络的每一个目标矢量都具有以下属性：

(1) size：该属性定义了每一个神经网络目标矢量的维数，属性值为只读变量，其数值等于第 i 层中神经元的个数(net.layers{i}.size)。

(2) userdata：该属性用于存储用户信息。

5. 阈值(biases)

阈值 net.biases 为 $N\times1$ 维的单元数组，第 i 个单元 net.biases{i}中存储了第 i 层神经元的阈值信息。网络中每一层神经元的阈值都具有以下属性：

(1) initFcn：该属性定义了阈值的初始化函数，其属性值为表示阈值初始化函数名称的字符串。

(2) learn：该属性定义了阈值在训练过程中是否进行调整，其属性值为 0 或 1。

(3) learnFcn：该属性定义了阈值的学习函数，其属性值为表示阈值学习函数名称的字符串。

(4) learnParam：该属性定义了阈值学习函数的参数，其属性值为各参数构成的结构体。

(5) size：该属性定义了每层神经网络的阈值个数，属性值为只读变量，其数值等于该层神经元的个数(net.layers{i}.size)。

(6) userdata：该属性用于存储用户信息。

6. 输入权值(inputWeights)

网络输入权值 net.inputWeights 为 $N\times Ni$ 维的单元数组，其中 net.inputWeights{i,j}为网络第 j 个输入到网络第 i 层的网络权值的属性。每一组网络输入权值都具有以下属性：

(1) delays：该属性定义了网络输入的各延迟拍数，其属性值是由 0 或正整数构成的行矢量，各输入层实际接收的是由网络输入的各个延迟构成的混合输入。

(2) initFcn：该属性定义了输入权值的初始化函数，其属性值为表示权值初始化函数名称的字符串。

(3) learn：该属性定义了输入权值在训练过程中是否进行调整，其属性值为 0 或 1。

(4) learnFcn：该属性定义了输入权值的学习函数，其属性值为表示权值学习函数名称的字符串。

(5) learnParam：该属性定义了权值学习函数的参数，其属性值为各参数构成的结构体。



(6) size：该属性定义了每个输入权值矩阵(net.IW{i,j})的大小，其属性值是一个具有两元素的行矢量，并且是只读变量，其中第一个元素等于该层神经元的个数(net.layers{i}.size)，第二个元素等于输入延迟矢量的长度(length(net.inputWeights{i,j}.delays))和输入矢量维数(net.inputs{j}.size)的乘积。

(7) userdata：该属性用于存储用户信息。

(8) weightFcn：该属性定义了输入权值的加权函数，其属性值为表示加权函数名称的字符串，该函数确定了输入和权值以何种方式组合在一起构成神经元的加权输入量。

7. 各层网络权值(layerWeights)

网络各层间的网络权值 net.layerWeights 为 $N \times N$ 维的单元数组，其中 net.layerWeights{i,j}为从网络第 j 层到第 i 层的网络权值的属性。网络权值具有以下属性：

(1) delays：该属性定义了网络第 j 层的输出送入到网络第 i 层之前要经过的各延迟拍数，其属性值是由 0 或正整数构成的行矢量。

(2) initFcn：该属性定义了权值的初始化函数，其属性值为表示权值初始化函数名称的字符串。

(3) learn：该属性定义了权值在训练过程中是否进行调整，其属性值为 0 或 1。

(4) learnFcn：该属性定义了权值的学习函数，其属性值为表示权值学习函数名称的字符串。

(5) learnParam：该属性定义了权值学习函数的参数，其属性值为各参数构成的结构体。

(6) size：该属性定义了权值矩阵(net.LW{i,j})的大小，其属性值是一个具有两元素的行矢量，并且为只读变量，其中第一个元素等于第 i 层神经元的个数(net.layers{i}.size)，第二个元素等于两层网络之间延迟矢量的长度(length(net.layerWeights{i,j}.delays))和第 j 层神经元个数(net.layers{j}.size)的乘积。

(7) userdata：该属性用于存储用户信息。

(8) weightFcn：该属性定义了加权函数，其属性值为表示加权函数名称的字符串。

3.2 基于工具箱函数的神经网络设计与分析

神经网络对象内封装了初始化函数、训练函数等网络属性，在建立了神经网络对象之后，只要正确设置对象的各个属性，就可以利用网络的使用函数完成对网络的初始化、训练和仿真。从神经网络对象的角度来说，MATLAB 中神经网络工具箱的结构可以用表 3-1 来加以概括。本节将根据该表的内容对网络设计和分析时常用的工具箱函数进行分类介绍。对于作为神经网络对象属性参数的函数，将说明如何把网络对象的属性设置为该函数。

为了方便读者在本书中查找工具箱函数的功能和使用方法，本书附录中列出了本章介绍的神经网络常用工具箱函数的索引表。



表 3-1 神经网络工具箱函数的构成

<table>
<tr><td rowspan="19">神经网络工具箱函数</td><td>网络建立函数</td><td colspan="2">自定义网络和各种常见网络的建立函数</td></tr>
<tr><td>网络使用函数</td><td>初始化、调整、训练和仿真函数</td><td>网络使用函数中的初始化函数 init 通过调用指定的某一种初始化函数来完成对网络的初始化。训练函数同理</td></tr>
<tr><td rowspan="3">网络初始化函数</td><td>网络初始化函数</td><td rowspan="12">这些函数都是在建立网络对象时需要设定的参数，在程序中不被直接调用，而是通过网络使用函数间接调用</td></tr>
<tr><td>层初始化函数</td></tr>
<tr><td>权值和阈值初始化函数</td></tr>
<tr><td rowspan="4">网络学习函数</td><td>训练和自适应调整函数</td></tr>
<tr><td>学习函数</td></tr>
<tr><td>性能函数及其导函数</td></tr>
<tr><td>线搜索函数</td></tr>
<tr><td rowspan="3">仿真函数</td><td>传递函数及其导函数</td></tr>
<tr><td>加权函数及其导函数</td></tr>
<tr><td>输入函数及其导函数</td></tr>
<tr><td rowspan="2">自组织网络专用函数</td><td>拓扑函数</td></tr>
<tr><td>距离函数</td></tr>
<tr><td rowspan="5">其他辅助函数</td><td colspan="2">数据预处理和后处理函数</td></tr>
<tr><td colspan="2">分析函数</td></tr>
<tr><td colspan="2">绘图函数</td></tr>
<tr><td colspan="2">网络计算函数</td></tr>
<tr><td colspan="2">矢量函数</td></tr>
</table>

3.2.1 神经网络的建立函数

表 3-2 列出了神经网络的建立函数，其中 network 函数用于建立由用户自定义的结构特殊的神经网络，这时用户要根据实际情况设定网络的属性参数。一般情况下，用户可利用 newp、newlin 和 newff 等网络建立函数方便地创建各种常用网络。

表 3-2 神经网络的建立函数

函数名称	功 能
network	建立一个自定义神经网络对象
newc	建立一个竞争层网络
newcf	建立一个前向级联 BP 网络
newelm	建立一个 Elman 反馈网络
newff	建立一个前向 BP 网络



续表

函数名称	功　　能
newfftd	建立一个有输入延迟的前向 BP 网络
newgrnn	设计一个广义回归网络
newhop	建立一个 Hopfield 反馈网络
newlin	建立一个线性神经网络
newlind	设计一个线性神经网络
newlvq	建立一个学习矢量量化网络
newp	建立一个感知器
newpnn	设计一个概率神经网络
newrb	设计一个径向基函数网络
newrbe	精确设计一个径向基函数网络
newsom	建立一个自组织映射网络

1. network

功能：建立一个自定义神经网络对象。

格式：

① net = network

② net = network (numInputs, numLayers, biasConnect, inputConnect, layerConnect, outputConnect, targetConnect)

说明：

该函数返回一个神经网络对象，函数调用形式②中的 numInputs 等输入量可确定神经网络结构的参数，其功能说明可参见 3.1 节中的介绍。

例 3.1 定义一个神经网络对象。

```
net = network
net =

    Neural Network object:

    architecture:

         numInputs: 0
         numLayers: 0
       biasConnect: [ ]
      inputConnect: [ ]
      layerConnect: [ ]
     outputConnect: [ ]
     targetConnect: [ ]

        numOutputs: 0  (read-only)
        numTargets: 0  (read-only)
```





```
        numInputDelays: 0   (read-only)
        numLayerDelays: 0   (read-only)

    subobject structures:

                          inputs: {0x1 cell} of inputs
                          layers: {0x1 cell} of layers
                         outputs: {1x0 cell} containing no outputs
                         targets: {1x0 cell} containing no targets
                          biases: {0x1 cell} containing no biases
            inputWeights: {0x0 cell} containing no input weights
            layerWeights: {0x0 cell} containing no layer weights

    functions:

            adaptFcn: (none)
             initFcn: (none)
          performFcn: (none)
            trainFcn: (none)

    parameters:

          adaptParam: (none)
           initParam: (none)
        performParam: (none)
          trainParam: (none)

    weight and bias values:
                   IW: {0x0 cell} containing no input weight matrices
                   LW: {0x0 cell} containing no layer weight matrices
                    b: {0x1 cell} containing no bias vectors

    other:

                 userdata: (user stuff)
```

上例在调用函数 network 时没有给出网络的结构参数，因此函数返回值 net 中的各属性参数均为函数提供的缺省值。

例 3.2 建立一个单输入三层前向级联网络，网络中每层神经元都具有阈值，网络的第一层为输入层，第三层为输出层，并且具有目标矢量，则其函数语句格式如下：

```
net = network (1, 3, [1; 1; 1], [1; 0; 0], [0 0 0; 1 0 0; 0 1 0], [0 0 1], [0 0 1])
```

其网络结构图如图 3.1 所示，图中的 IF1 和 TF1 等模块分别表示输入函数和传递函数。



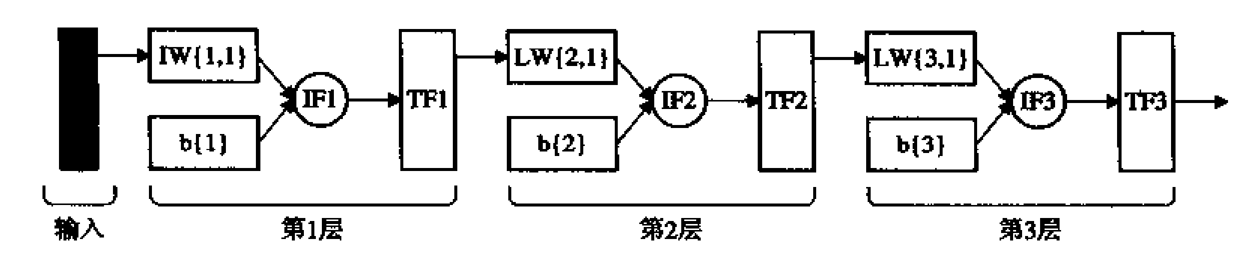


图 3.1 三层前向神经网络结构图

2. newc

功能：建立一个竞争层网络。

格式：

① net = newc

② net = newc (PR, S, KLR, CLR)

说明：

竞争层网络由单一的竞争层构成，主要用于解决分类问题。竞争层的加权函数为 negdist，输入函数为 netsum，传递函数为 compet。神经元权值和阈值的初始化函数分别为 midpoint 和 initcon，网络的自适应调整函数和训练函数分别为 trains 和 trainr，权值和阈值的学习函数分别为 learnk 和 learncon。

函数的调用形式①返回一个没有定义结构的空对象，并显示图形用户界面函数 nntool 的帮助文字。

函数的调用形式②返回竞争层网络对象 net，此时函数的自变量 PR 为表示网络输入矢量取值范围的矩阵[Pmin Pmax]；S 为竞争层神经元个数，也是网络输入矢量的分类数；KLR 为权值学习速率，缺省值为 0.01；CLR 为阈值学习速率，缺省值为 0.001。

例 3.3 观察调用形式①返回的结果。

```
net = newc
To create a neural network using a graphical user interface:
1. Use the command "nntool" to open the Neural Network Manager.
2. Click "New Network" to open the dialog for creating networks.
You can then import or define data and train your network in the GUI
or export your network to the command line for training.

net =

      [ ]
```

例 3.4 利用竞争层网络把下列二维矢量分为两类。

```
P=[0.1    0.8    0.1    0.9    0.2    0.8;
   0.2    0.9    0.1    0.8    0.1    0.8];
```

该二维输入矢量的取值范围矩阵为

```
PR=[0 1; 0 1]
```



若把这些矢量分为两类，新建的竞争层应由两个神经元构成，其网络结构如图 3.2 所示。

net = newc ([0 1; 0 1], 2);

对网络进行训练，则语句调用格式为

net = train (net, P);

网络的仿真结果为

Y = sim (net, P)

Y =

(2, 1) 1
(1, 2) 1
(2, 3) 1
(1, 4) 1
(2, 5) 1
(1, 6) 1

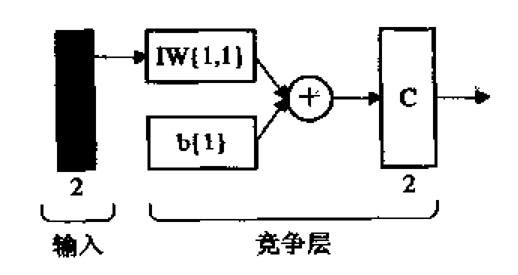


图 3.2 竞争层神经网络结构图

其中，Y是一个稀疏矩阵，其第一行的2、4、6列元素和第二行的1、3、5列元素均为1，即P中第 2、4、6个输入矢量为第一类，其余三个矢量属于第二类。为了便于观察结果，可将Y转化为下标矢量形式，即

Yc = vec2ind (Y)

Yc =

2 1 2 1 2 1

参见：sim、init、adapt、train、trains 和 trainr 函数。

3. newcf

功能：建立一个前向级联 BP 网络。

格式：

① net = newcf

② net = newcf (PR, [S1 S2⋯SN], {TF1 TF2⋯TFN}, BTF, BLF, PF)

说明：

该函数可以建立一个 N 层前向级联网络。网络各层的加权函数为 dotprod，输入函数为 netsum，各层传递函数由用户确定。各神经元权值和阈值的初始化函数为 initnw，网络的自适应调整函数为 trains，并根据指定的学习函数对权值和阈值进行更新，网络的训练函数由用户指定。

函数调用形式①参见 newc 函数。函数的调用形式②返回前向级联网络对象 net，此时函数的自变量 PR 为表示网络输入矢量取值范围的矩阵[Pmin Pmax]；S1～SN 为各层的神经元个数；TF1～TFN 用于指定各层神经元的传递函数，可以是 tansig、logsig 或 purelin 等可微传递函数，缺省值为'tansig'；BTF 用于指定网络的训练函数，可以是 trainlm、trainbfg、trainrp 或 traingd 等采用 BP 算法的训练函数，缺省值为'trainlm'；BLF 用于指定权值和阈值的学习函数，缺省值为'learngdm'，也可以是基于其他 BP 算法的学习函数；PF 用于指定网络的性能函数，可以是 mse、msereg 等可微性能函数，缺省值为'mse'。



例 3.5 假设输入和目标矢量矩阵分别为

P = [8 7 6 5 4 3 2 1 0];

T = [0 1 2 3 2 1 2 3 2];

建立一个两层前向级联网络，其中第一层有六个神经元，采用tansig传递函数，输出层神经元的传递函数为purelin，其他参数均采用缺省值，其网络结构如图3.3所示。

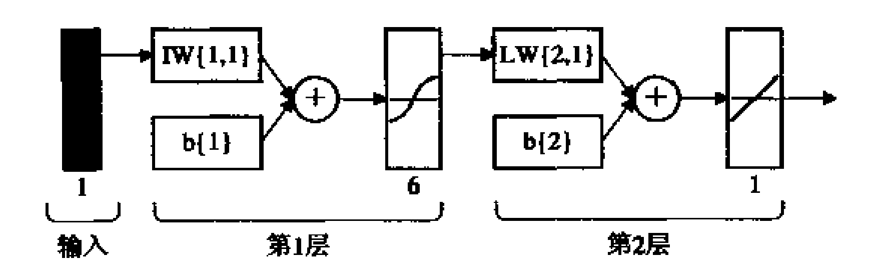


图 3.3 两层级联前向神经网络的结构图

net = newcf ([0 8], [6 1], {'tansig' 'purelin'});

对网络进行训练和仿真：

net = train (net、P、T);

Y = sim (net、P)

训练和仿真结果为

Y =

-0.0000 1.0000 2.0000 3.0000 2.0000 1.0000 2.0000 3.0000 2.0000

参见：newff、newelm、sim、init、adapt、train 和 trains 函数。

4. newelm

功能：建立一个 Elman 反馈网络。

格式：

① net = newelm

② net = newelm (PR, [S1 S2…SN], { TF1 TF2…TFN }, BTF, BLF, PF)

说明：

该函数可以建立一个 N 层 Elman 网络。网络中各层依次级联，除最后一层外网络的每层都具有自反馈，最后一层为网络的输出层。网络各层的加权函数为 dotprod，输入函数为 netsum，各层传递函数由用户设定。各神经元权值和阈值的初始化函数为 initnw，网络的自适应调整函数为 trains，并根据指定的学习函数对权值和阈值进行更新，网络的训练函数由用户指定。

函数调用形式①参见 newc 函数。函数的调用形式②返回 Elman 网络对象 net，此时函数输入 PR 为表示网络输入矢量取值范围的矩阵[Pmin Pmax]；S1～SN 为各层神经元的个数；TF1～TFN 用于指定各层神经元的传递函数，缺省值为'tansig'，也可以设为 logsig、purelin 等可微传递函数；BTF 用于指定训练函数，缺省值为'traingdx'，可以设为 trainbfg 或 traingd 等采用 BP 算法的训练函数，但不宜使用 trainlm、trainrp 等采用大学习步长的函数；BLF 用于指定权值和阈值的学习函数，缺省值为'learngdm'，也可以设为基于其他 BP



算法的学习函数；PF 用于指定性能函数，缺省值为'mse'，也可以设为 msereg 等可微性能函数。

例 3.6 假设输入和目标序列分别为

Pseq = {0, 0, 1, 1, 0, 1, 0, 0, 1};

Tseq = {0, 1, 0, 1, 0, 0, 0, 1, 0};

建立一个两层Elman BP网络，其中第一层有六个神经元，传递函数为tansig，该层具有自反馈，输出层神经元的传递函数为logsig，其他参数均采用缺省值，其网络结构如图3.4所示。

```
net = newelm ( [0   1], [6   1], {'tansig'   'logsig'} )
```

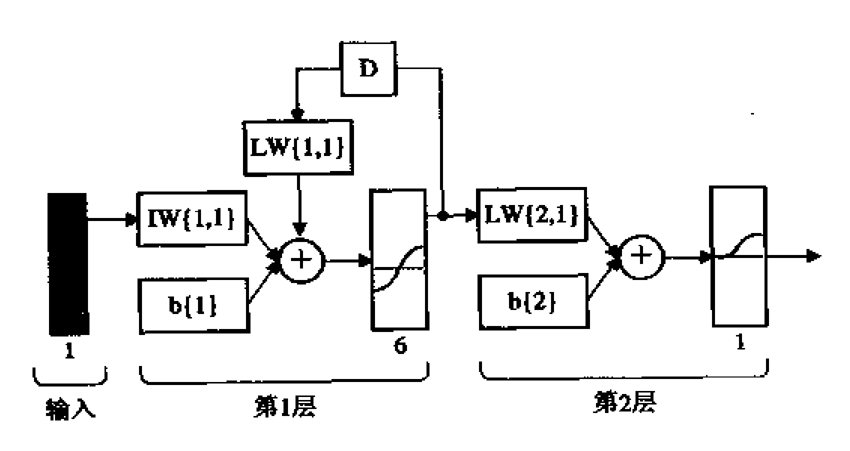


图 3.4 两层 Elman BP 网络结构图

然后对网络进行训练和仿真：

```
net.trainParam.epochs=500;
net = train ( net, Pseq, Tseq );
Y = sim ( net, Pseq )
```

训练和仿真结果为

```
Y =
    [0.0001] [0.9993] [0.0005] [0.9985] [0.0001] [0.0011] [0.0013] [0.9997] [0.0005]
```

参见：newff、newcf、sim、init、adapt、train 和 trains 函数。

5. newff

功能：建立一个前向 BP 网络。

格式：

① net = newff

② net = newff (PR, [S1 S2⋯SN], { TF1 TF2⋯TFN }, BTF, BLF, PF)

说明：

该函数可以建立一个 N 层前向 BP 网络。网络各层的加权函数为 dotprod，输入函数为 netsum，各层传递函数由用户设定。各神经元权值和阈值的初始化函数为 initnw，网络的自适应调整函数为 trains，并根据指定的学习函数对权值和阈值进行更新，网络的训练函数由用户指定。



函数调用形式①参见 newc 函数。函数的调用形式②返回 BP 网络对象 net，此时函数的自变量 PR 为表示网络输入矢量取值范围的矩阵[Pmin Pmax]；S1～SN 为各层神经元的个数；TF1～TFN 用于指定各层神经元的传递函数，缺省值为'tansig'，也可设为 logsig 或 purelin 等可微函数；BTF 用于指定网络的训练函数，可以设为 trainlm、trainbfg、trainrp 或 traingd 等采用 BP 算法的训练函数，缺省值为'trainlm'；BLF 用于指定权值和阈值的学习函数，缺省值为'learngdm'，也可以是其他采用 BP 算法的学习函数；PF 用于指定网络的性能函数，可设为 mse、msereg 等可微性能函数，缺省值为'mse'。

例 3.7　对以下样本数据，采用 newff 建立前向神经网络，其网络结构如图 3.3 所示。

```
P = [8  7  6  5  4  3  2  1  0];
T = [0  1  2  3  2  1  2  3  2];
net = newff ( [ 0   8 ], [ 6   1 ], { 'tansig'   'purelin' } );
```

对网络进行训练和仿真：

```
net = train ( net, P, T );
Y = sim ( net, P )
```

训练和仿真结果为

```
Y =
    0.0000  1.0000  2.0000  3.0000  2.0000  1.0000  2.0000  3.0000  2.0000
```

参见：newcf、newelm、sim、init、adapt、train 和 trains 函数。

6. newfftd

功能：建立一个具有输入延迟的前向 BP 网络。

格式：

① net = newfftd

② net = newfftd (PR, ID, [S1 S2…SN], { TF1 TF2…TFN }, BTF, BLF, PF)

说明：

该函数用于建立一个 N 层前向 BP 网络，网络输入经由指定延迟后进入网络的第一层。网络各层的加权函数为 dotprod，输入函数为 netsum，各层传递函数由用户设定。各神经元权值和阈值的初始化函数为 initnw，网络的自适应调整函数为 trains，并根据指定的学习函数对权值和阈值进行更新，网络的训练函数由用户指定。

函数调用形式①参见 newc 函数。函数的调用形式②返回具有输入延迟的 BP 网络对象 net，此时函数的输入 PR 为表示网络输入矢量取值范围的矩阵[Pmin Pmax]；ID 为输入延迟矢量；S1～SN 为各层的神经元个数；TF1～TFN 指定各层神经元的传递函数，可以是 tansig、logsig 或 purelin 等可微函数，缺省值为'tansig'；BTF 用于指定网络的训练函数，可以是 trainlm、trainbfg、trainrp 等采用 BP 算法的训练函数，缺省值为'trainlm'；BLF 用于指定权值和阈值的学习函数，缺省值为'learngdm'，也可以是基于其他 BP 算法的学习函数；PF 用于指定网络的性能函数，缺省值为'mse'，也可设为 msereg 等可微性能函数。

例 3.8　假设输入和目标序列分别为

```
P = {1  0  0  1  1  0  1  0 };
T = {1 -1  0  1  0 -1  1 -1 };
```



建立一个两层网络，网络的第一层有三个神经元，该层输入由网络输入及其一拍延迟量构成，神经元传递函数为tansig，输出层神经元传递函数为purelin，其网络结构如图3.5所示。

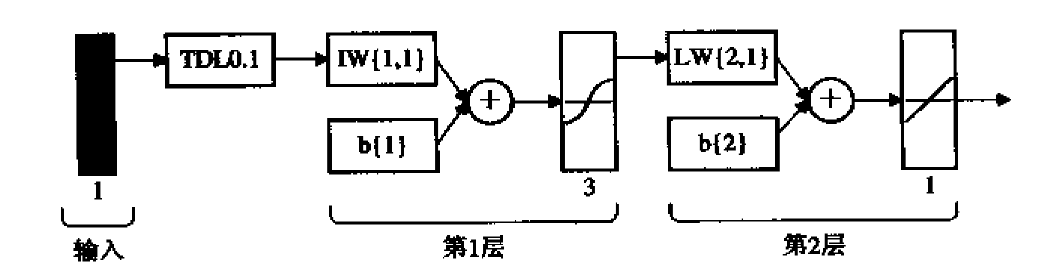


图 3.5　具有一拍输入延迟的两层前向网络

```
net = newfftd ( [0   1], [0   1], [3   1], {'tansig'   'purelin'} );
```

对网络进行训练和仿真：

```
net.trainParam.epochs=50;
net = train ( net, P, T );
Y = sim ( net, P )
```

训练和仿真结果为

```
Y =
    [1.0000]  [-1.0000]  [0.0000]  [1.0000]  [0.0000]  [-1.0000]  [1.0000]  [-1.0000]
```

参见：newcf、newelm、sim、init、adapt、train 和 trains 函数。

7. newgrnn

功能：设计一个广义回归网络。

格式：

① net = newgrnn

② net = newgrnn (P, T, spread)

说明：

广义回归网络是径向基函数网络的变形，主要用于解决函数逼近问题，本函数可以快速设计一个广义回归网络。广义回归网络是一个两层神经网络，第一层采用径向基神经元，传递函数为 radbas，加权函数为 dist，输入函数为 netprod；第二层神经元的传递函数为 purelin，加权函数为 normprod，输入函数为 netsum。

函数调用形式①参见 newc 函数。函数的调用形式②返回广义回归网络对象 net，此时函数的输入 P、T 分别为网络输入样本矢量和输出目标矢量构成的矩阵；spread 为扩展常数，spread 越大，则函数越平滑，当 spread 小于输入矢量间的典型距离时，网络拟合的函数将非常逼近于样本矢量数据，spread 的缺省值为 1。newgrnn 函数把第一层神经元权值设置为输入 P 的转置矩阵 P'，阈值设置为 0.8326/spread；第二层神经元权值设置为目标矢量 T。



例 3.9　假设输入和目标矢量的矩阵分别为

```
P = [0    1    2    3    4 ];
T = [-1   -3.1   -5.4   -4.3   -2 ];
```

设计一个广义回归网络，其网络结构如图 3.6 所示。

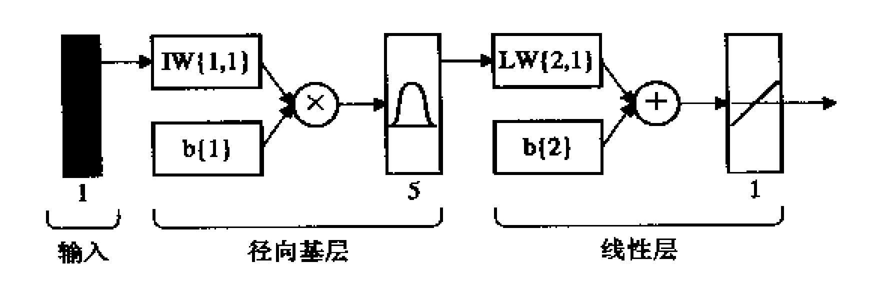


图 3.6　广义回归网络结构图

```
net = newgrnn ( P, T );
```

仿真结果为

```
Y= sim ( net, P )
Y=
   -1.8511   -3.1837   -4.3706   -3.9699   -2.8723
```

参见：sim、newrb、newrbe 和 newpnn 函数。

8. newhop

功能：建立一个 Hopfield 反馈网络。

格式：

① net = newhop

② net = newhop (T)

说明：

Hopfield 网络主要用于解决联想记忆问题，该网络由一层神经元组成，并且具有自反馈。网络神经元的加权函数为 dotprod，输入函数为 netsum，神经元传递函数为 satlins。Hopfield 网络运行稳定后的输出为用户预先提供的目标矢量，即网络的稳定点。

函数调用形式①参见 newc 函数。函数的调用形式②返回 Hopfield 网络对象 net，此时函数输入 T 为目标矢量组成的矩阵，目标矢量中的元素必须是 +1 或 −1。

例 3.10　假定目标矢量为

```
T = [-1 -1 1 1; 1 1 -1 -1]';
```

建立一个Hopfield网络，其网络结构如图3.7所示。

```
net = newhop ( T );
Ai = {[-0.9;   -0.8;   0.7; 0.7]};
Y= sim ( net, {1, 2}, {}, Ai );
```

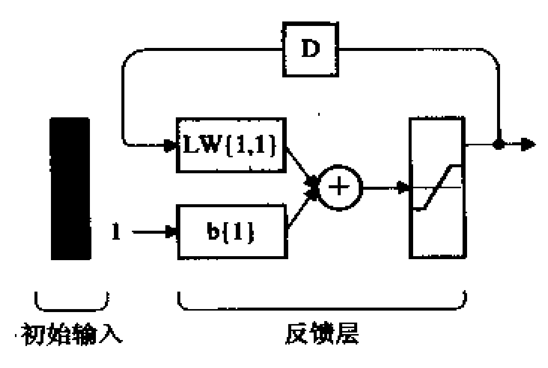


图 3.7　Hopfield 网络结构图



对下面的矢量进行联想记忆，并让网络工作两个时间步长，运行结果为

```
Y{1, 1}'
ans =
  -0.9283  -0.9060   0.8837   0.8837
Y{1, 2}'
ans =
    -1      -1       1       1
```

可见网络在工作两个时间步长后就已达到稳定状态。

参见：sim 和 satlins 函数。

9. newlin

功能：建立一个线性神经网络。

格式：

① net = newlin

② net = newlin (PR, S, ID, LR)

③ net = newlin (PR, S, 0, P)

说明：

线性神经网络主要用于自适应滤波器设计和信号预测，该网络由一层神经元组成，神经元的加权函数为 dotprod，输入函数为 netsum，传递函数为 purelin。神经元的权值和阈值初始化函数为 initzero，自适应调整函数和训练函数分别为 trains 和 trainb， 它们都使用学习函数 learnwh 对权值和阈值进行调整，性能函数为 mse。

函数调用形式①参见 newc 函数。函数的调用形式②返回线性神经网络对象 net，此时函数输入 PR 为表示网络输入矢量取值范围的矩阵[Pmin Pmax]；S 为输出矢量的维数；ID 为输入延迟矢量，缺省值为[0]；LR 为学习速率，缺省值为 0.01。

在调用形式③中，自变量 PR 和 S 的含义如调用形式②，P 为网络输入矢量组成的矩阵。这时函数除建立一个线性神经网络对象外，还会根据输入矢量 P 确定最大的稳定学习速率。

例 3.11 假设输入和目标矢量矩阵为

```
P = [0    1    2    3    4 ];
T = [-1  -3.1  -5.4  -7.3  -8.2 ];
```

建立一个单神经元线性神经网络，其网络结构如图 3.8 所示。

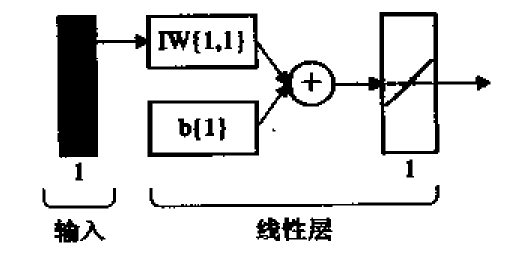


图 3.8 线性神经网络结构图

```
net = newlin ( [0   5], 1 );
```

对网络进行训练和仿真

```
net = train ( net, P, T );
Y = sim ( net, P )
Y =
     -1.1557   -3.0593   -4.9629   -6.8665   -8.7701
```

参见：newlind、sim、init、adapt、train、trains 和 trainb 函数。



10. newlind

功能：设计一个线性神经网络。

格式：

① net = newlind

② net = newlind (P, T, Pi)

说明：

该函数根据用户提供的输入矢量和目标矢量设计线性神经网络，并使得网络输出矢量和目标矢量间达到最小均方误差。该函数利用最小二乘法求解网络权值和阈值。

函数调用形式①参见 newc 函数。函数调用形式②返回线性神经网络对象 net，此时函数自变量 P 为网络的输入矢量；T 为目标矢量；Pi 为初始输入延迟条件，缺省值为[ ]。函数输入可以是矩阵或单元数组，单元数组的形式适用于多输入和输入有延迟的情况。

例 3.12　根据以下样本数据设计一个线性神经网络，其网络结构如图 3.8 所示。

```
P = [0     1     2     3     4 ];
T = [-1   -3.1  -5.4  -7.3  -8.2 ];
net = newlind ( P, T );
```

仿真结果为

```
Y = sim ( net, P )
Y =
    -1.2800    -3.1400    -5.0000
    -6.8600    -8.7200
```

参见：newlin 和 sim 函数。

11. newlvq

功能：建立一个学习矢量量化网络。

格式：

① net = newlvq

② net = newlvq (PR, S1, PC, LR, LF)

说明：

学习矢量量化网络主要用于解决分类问题。该网络由两层神经元组成，网络的第一层为竞争层；第二层中的每个神经元只与竞争层中的部分神经元相连，网络中的每个神经元都没有阈值。网络第一层神经元的传递函数为 compet，加权函数为 negdist，权值初始化函数为 midpoint；网络第二层神经元的传递函数为 purelin，加权函数为 dotprod ，权值都为 1。网络使用 trains 和 trainr 函数进行自适应调整和训练，这两个函数使用指定的学习函数对竞争层神经元的权值进行修正。

函数调用形式①参见 newc 函数。函数的调用形式②返回学习矢量量化网络对象 net，函数的输入 PR 为表示网络输入矢量取值范围的矩阵[Pmin Pmax]；S1 为竞争层神经元的个数；PC 为 S^2（直接由 PC 的维数确定）维矢量，其中 S^2 为网络输入矢量的类别数，也是输出层的神经元个数，PC(i)为网络输入矢量中第 i 类矢量所占的比例，即输出层中和竞争层第 i 个神经元相连的神经元所占的比例；LR 为学习速率，缺省值为 0.01；LF 为学习函



数，可以是 learnlv1 或 learnlv2 函数，缺省值为'learnlv1'，网络只有先使用 learnlv1 函数进行学习后，才能再用 learnlv2 进行学习。

例 3.13 假定要将下列样本矢量分为两类，每类所占的比例均为 0.5：

```
P = [3  -2  -1  2  2  -1 ;
     2  -1  -1  2  3  -2];
T = [1   2   2  1  1   2 ];
```

建立学习矢量量化网络，网络竞争层由四个神经元组成，其网络结构如图3.9所示。

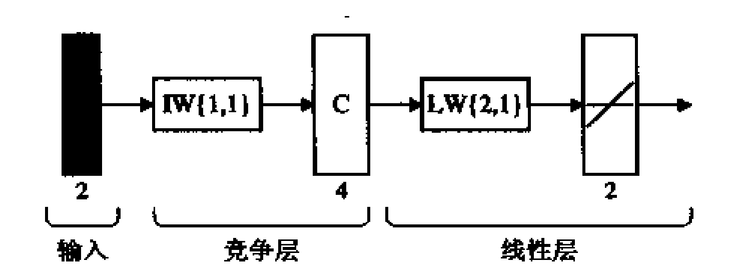


图 3.9 学习矢量量化网络结构图

```
net = newlvq ( [-2  3;  -2  3], 4, [0.5  0.5] );
```

下标矢量T必须转化为单值矢量组，才能作为目标矢量使用：

```
T1 = ind2vec ( T );
net = train ( net, P, T1 );
```

仿真结果为

```
Y = vec2ind ( sim ( net, P ) )
Y =
    1    2    2    1    1    2
```

参见：sim、init、adapt、train、trains、trainr、learnlv1 和 learnlv2 函数。

12. newp

功能：建立一个感知器。

格式：

① net = newp

② net = newp (PR, S, TF, LF)

说明：

感知器用于解决线性可分问题。该网络由一层神经元构成，神经元的加权函数为 dotprod，输入函数为 netsum，传递函数可以是 hardlim 或 hardlims，网络权值和阈值的初始化函数为 initzero，采用训练函数 trains 和 trainc 对网络进行自适应调整和训练，学习函数可以是 learnp 或 learnpn，性能函数为 mae。

函数调用形式①参见 newc 函数。函数的调用形式②返回感知器网络对象 net，此时函数的输入 PR 为表示网络输入矢量取值范围的矩阵[Pmin Pmax]；S 为神经元个数；TF 为传递函数，缺省值为'hardlim'；LF 为学习函数，缺省值为'learnp'。

例 3.14 根据下列样本矢量和目标矢量：



```
P = [3  -2   -1   2   2   -1 ;
     2  -1   -1   2   3   -2];
T = [1   0    0   1   1    0 ];
```

建立一个感知器网络，该网络由一个神经元构成，其网络结构如图3.10 所示。

```
net = newp ( [-2  3; -2  3 ], 1 );
```

训练和仿真结果为

```
net = train ( net, P, T );
Y = sim( net, P )
Y =
     1     0     0     1     1     0
```



图 3.10 感知器网络结构图

参见：sim、init、adapt、train、hardlim、hardlims、learnp、learnpn、trains 和 trainc 函数。

13. newpnn

功能：设计一个概率神经网络。

格式：

① net = newpnn

② net = newpnn (P, T, spread)

说明：

概率神经网络是径向基函数网络的变形，主要用于解决分类问题。概率神经网络由两层神经元构成，只有第一层神经元具有阈值。网络的第一层采用径向基神经元，传递函数为 radbas，加权函数为 dist，输入函数为 netprod；第二层神经元采用竞争传递函数 compet，加权函数为 dotprod。

函数调用形式①参见 newc 函数。函数的调用形式②返回概率神经网络 net，函数输入 P、T 分别为输入样本矢量和目标输出矢量构成的矩阵；spread 为扩展常数，缺省值为 0.1。newpnn 函数把第一层神经元的权值设置为网络输入矩阵 P 的转置矩阵 P'，阈值设置为 0.8326/ spread；第二层神经元的权值设置为网络的目标矢量 T。

例 3.15 根据下列样本矢量和目标矢量：

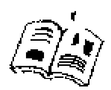
```
P = [3  -2   -1   2   2   -1 ;
     2  -1   -1   2   3   -2];
T = [1   2    2   1   1    2 ];
T1 = ind2vec ( T );
```

设计一个概率神经网络，其网络结构如图3.11所示。

```
net = newpnn ( P, T1 );
```

仿真结果为

```
Yc = vec2ind ( sim ( net, P ) )
Yc =
     1     2     2     1     1     2
```



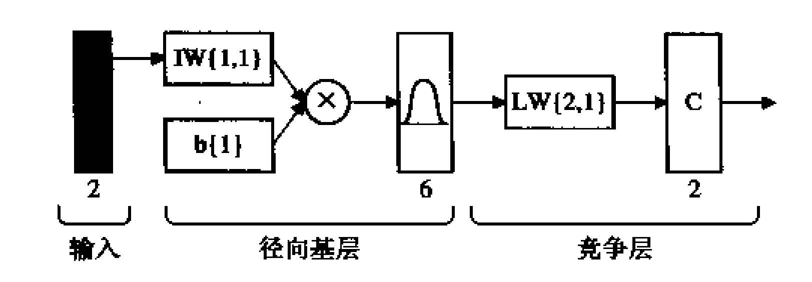


图 3.11　概率神经网络结构图

参见： sim、newrb、newrbe、newgrnn、ind2vec 和 vec2ind 函数。

14. newrb

功能： 设计一个径向基函数网络。

格式：

① net = newrb

② net = newrb (P, T, goal, spread, MN, DF)

说明：

径向基函数网络主要用于解决函数逼近问题。径向基函数网络由两层神经元构成，网络的第一层为径向基层，神经元传递函数为 radbas，加权函数为 dist，输入函数为 netprod；第二层神经元的传递函数为纯线性函数 purelin，加权函数为 dotprod，输入函数为 netsum。

函数调用形式①参见 newc 函数。函数的调用形式②返回径向基函数网络对象 net，此时函数的输入 P 为网络输入样本矢量；T 为目标矢量；goal 为网络的均方误差性能指标；spread 为扩展常数，缺省值为 1.0；MN 为神经元个数最大值，缺省值为输入样本矢量的个数 ；DF 为训练过程的显示频率，缺省值为 25。

该函数利用迭代方法建立网络，开始时网络径向基层的神经元个数为零，然后每迭代一次，径向基层就添加一个神经元。在每次迭代中，网络首先进行仿真并找到对应于最大输出误差的输入样本矢量，然后径向基层添加一个神经元并把权值设为该输入矢量，最后再修改线性层的权值以达到最小误差。

例 3.16　根据以下数据，设计一个径向基函数神经网络：

```
P = [0     1     2     3     4 ];
T = [-1   -3.1  -5.4  -4.3  -2 ];
net = newrb ( P, T );
```

网络结构如图3.12所示。仿真结果为

```
Y = sim ( net, P )
Y =
    -1.0000    -3.1000    -5.4000    -4.3000    -2.0000
```

参见： sim、newrbe、newgrnn 和 newpnn 函数。



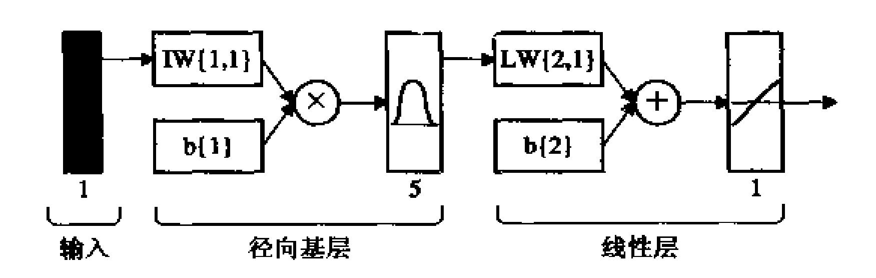


图 3.12　径向基函数网络结构图

15. newrbe

功能：精确设计一个径向基函数网络。

格式：

① net = newrbe

② net = newrbe (P, T，spread)

说明：

该函数可以精确设计一个径向基函数网络，使得网络对输入样本矢量的响应输出和目标矢量相同。

函数调用形式①参见 newc 函数。函数的调用形式②返回径向基函数网络对象 net，此时函数的输入 P 和 T 分别为输入样本矢量矩阵和目标矢量矩阵；spread 为扩展常数，缺省值为 1.0。newrbe 函数把径向基层的神经元权值设置为网络输入矩阵 P 的转置矩阵 P′，阈值设置为 0.8326/spread；第二层神经元的权值 W{2,1}和阈值 b{2}满足下式：

[W{2,1} b{2}] * [A{1}; ones] = T

其中，A{1}为第一层的输出。

例 3.17　对以下样本数据精确设计一个径向基函数网络，其网络结构图如图 3.12 所示。

```
P = [ 0     1     2     3     4 ];
T = [-1   -3.1  -5.4  -4.3  -2 ];
net = newrbe ( P, T );
```

仿真结果为

```
Y = sim ( net, P )
Y =
    -1.0000    -3.1000    -5.4000    -4.3000    -2.0000
```

参见：sim、newrb、newgrnn 和 newpnn 函数。

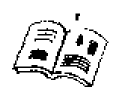
16. newsom

功能：建立一个自组织映射网络。

格式：

① net = newsom

② net = newsom (PR，[D1, D2, …], TFCN，DFCN，OLR，OSTEPS，TLR，TND)



说明：

自组织映射网络常用于解决分类问题。该网络由一层神经元构成，神经元按照用户指定的拓扑结构在多维空间中进行排列，神经元没有阈值。神经元的加权函数为 negdist，输入函数为 netsum，传递函数为竞争函数 compet，权值初始化函数为 midpoint，自适应调整函数和训练函数分别为 trains 和 trainr，学习函数为 learnsom。网络的训练分为排列和调整两个阶段进行。

函数调用形式①参见 newc 函数。函数的调用形式②返回自组织映射网络对象 net，此时函数的输入 PR 为表示网络输入矢量取值范围的矩阵[Pmin Pmax]；Di 为神经元在多维空间中排列时各维的个数，缺省值为[5，8]；TFCN 为拓扑函数，缺省值为'hextop'；DFCN 为距离函数，缺省值为'linkdist'；OLR 为排列阶段学习速率，缺省值为 0.9；OSTEPS 为排列阶段学习次数，缺省值为 1000；TLR 为调整阶段学习速率，缺省值为 0.02；TND 为调整阶段邻域半径，缺省值为 1。

例 3.18 假定要把下列样本矢量分为四类

```
P = [3  -2   1   2  -2  -1  -2   1;
     2  -1  -1   2   3  -2   2   3];
```

则需要建立一个四神经元的自组织映射网络，其网络结构如图3.13所示。

```
net = newsom ( [-2  3; -2  3], [2  2] );
```

网络训练和仿真结果为

```
net = train ( net, P );
Y = vec2ind ( sim ( net, P ) )
Y =
    1    4    2    1    3    4    3    1
```

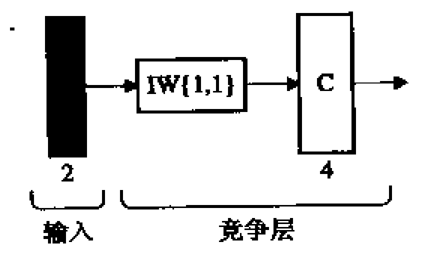


图 3.13 自组织映射网络结构图

参见：sim、init、adapt 和 train 函数。

3.2.2 神经网络的使用函数

表 3-3 列出了神经网络的基本使用函数，它们可分别用来对神经网络进行训练、显示、初始化和仿真，调用这些函数可以完成对神经网络对象的基本操作。

表 3-3 神经网络的使用函数

函数名称	功能
Adapt	对神经网络进行自适应调整
Disp	显示神经网络对象的属性
Display	显示神经网络对象的名称和属性
Init	对神经网络的参数进行初始化
Revert	将神经网络的权值和阈值重新设置为初始值
train	对神经网络进行训练
Sim	对神经网络进行仿真

1. adapt

功能：对神经网络进行自适应调整。

格式：

[net, Y, E, Pf, Af, TR] = adapt (net, P, T, Pi, Ai)

说明：

对于神经网络的每一个输入，adapt 函数将计算网络的输出及误差，然后调用自适应调整函数 net.adaptFcn，并根据自适应调整函数参数 net.adaptParam 对网络中的权值及阈值进行调整。

adapt 函数的输入 net 为神经网络对象，P 为输入矢量，T 为目标矢量，Pi 为输入延迟的初始状态，Ai 为层延迟的初始状态，其中，P 和 T 中可以是多个时刻的输入和目标矢量。在函数返回值中，net 为更新后的神经网络对象，Y 为网络输出，E 为输出和目标之间的误差，Pf 为调整终止时的输入延迟状态，Af 为调整终止时的层延迟状态，TR 中记录了训练次数和训练过程中网络性能的变化情况。函数中的 Pi、Ai、Pf 和 Af 仅用于具有输入延迟和层延迟的网络。P、T、Pi、Ai、Y、E、Pf 和 Af 等输入和输出量可以是单元数组或矩阵，但一般情况下都采用单元数组的形式，只有当 P 和 T 中仅包含一个时间步长的输入和目标矢量时，才采用矩阵形式。

示例参见例 3.19。

参见：sim、init、train 和 revert 函数。

2. disp 和 display

功能：显示神经网络对象的属性。

格式：

① disp (net)

② display (net)

说明：

这两个函数都用于显示神经网络对象的属性，唯一不同之处在于 display 函数还要显示神经网络对象的名称，而 disp 则不显示。

参见：adapt、sim、init 和 train 函数。

3. init

功能：对神经网络的参数进行初始化。

格式：

net = init (net)

说明：

该函数通过调用初始化函数 net.initFcn，并根据初始化函数参数 net.initParam 对网络权值和阈值进行初始化。

示例参见例 3.19。

参见：adapt、sim、train、initlay、initnw、initwb、rands 和 revert 函数。



4. revert

功能：该函数将网络中的权值和阈值恢复为初始值。

格式：

net = revert (net)

说明：

该函数将网络中的权值和阈值恢复为最近一次初始化时产生的初始数值，如果网络结构在初始化后发生了变化，权值和阈值将不能恢复，这时将把它们都设置为 0。

示例参见例 3.19。

参见：adapt、sim、train 和 init 函数。

5. train

功能：对神经网络进行训练。

格式：

[net, TR, Y, E, Pf, Af] = train (net, P, T, Pi, Ai, VV, TV)

说明：

train 函数通过调用网络训练函数 net.trainFcn，并根据训练函数参数 net.trainParam 对网络进行训练。

其中，net 为神经网络对象，P 为网络输入，T 为目标矢量，Pi 为输入延迟的初始状态，Ai 为层延迟的初始状态，VV 为验证矢量，TV 为测试矢量。在函数返回值中，TR 为训练记录，Y 为网络输出，E 为输出和目标矢量之间的误差，Pf 为训练终止时的输入延迟状态，Af 为训练终止时的层延迟状态。该函数中的 P、T、Pi、Ai、VV、TV、Y、E、Pf 和 Af 各参量可以是单元数组或矩阵，但一般情况下都采用单元数组的形式。

验证矢量 VV 和测试矢量 TV 用于提高和检验网络的推广能力。当网络的进一步训练破坏了网络对验证矢量 VV 的推广能力时，网络的训练将提前停止；测试矢量则用于检测网络的推广能力。

示例参见例 3.19。

参见：adapt、sim、init 和 revert 函数。

6. sim

功能：对神经网络进行仿真。

格式：

① [Y, Pf, Af, E, perf] = sim (net, P, Pi, Ai, T)

② [Y, Pf, Af, E, perf] = sim (net, {Q TS}, Pi, Ai, T)

③ [Y, Pf, Af, E, perf] = sim (net, Q, Pi, Ai, T)

说明：

sim 函数用于实现神经网络的仿真。

在 sim 函数的调用形式①中，输入 net 为神经网络对象，P 为网络输入，Pi 为输入延迟的初始状态，Ai 为层延迟的初始状态，T 为目标矢量。在函数返回值中，Y 为网络输出，Pf 为训练终止时的输入延迟状态，Af 为训练终止时的层延迟状态，E 为输出和目标矢

量之间的误差，perf 为网络的性能值。该函数中的 P、T、Pi、Ai、Y、E、Pf 和 Af 等参量可以是单元数组或矩阵。sim 函数的调用形式②③用于没有输入的网络，其中，Q 为批处理数据的个数，TS 为网络仿真的时间步数。

例 3.19　假定网络的输入和目标序列分别为

P = {0 -1 1 1 0 -1 1 0 0 1 1 0 -1 -1 1 1 1 0 -1};

T = {0 -1 0 2 1 -1 0 1 0 1 2 1 -1 -2 0 2 2 1 0};

这里的P和T都是单元数组。建立一个具有输入延迟的线性神经网络对P和T进行学习：

```
net = newlin ( [-1 1], 1, [0 1], 0.005 );
```

该网络输入的数值范围是[-1 1]，网络由一个神经元构成，该神经元的输入为当前网络输入及其一拍延迟量，网络学习速率为0.005，其网络结构如图3.14所示。

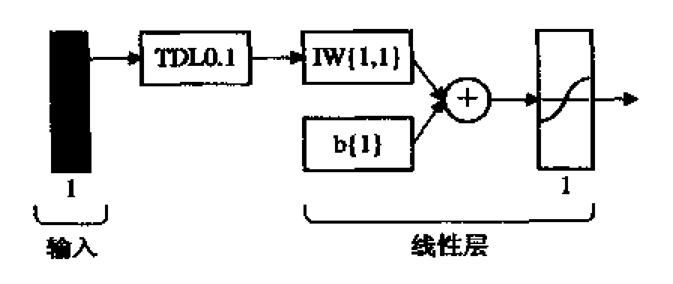


图 3.14　带有输入延迟的线性网络结构图

首先对网络进行初始化，并查看网络初始化后网络的权值和阈值：

```
net = init ( net );
net.IW {1, 1}
ans =
        0        0
net.b {1}
ans =
        0
```

然后对网络进行自适应调整：

```
[ net, y, e, pf, af ] = adapt ( net, P, T );
```

经过该轮自适应调整后，神经元的权值和阈值变为

```
net.IW {1, 1}
ans =
        0.0677      0.0675
net.b {1}
ans =
        0.0377
```

这时使用恢复函数，将权值和阈值重新设置为初始值：

```
net = revert ( net );
net.IW {1, 1}
ans =
        0        0
net.b {1}
ans =
        0
```

可见网络中的权值和阈值已经恢复为初始值。



最后，利用 train 函数对网络训练 100 次：

```
net.trainParam.epochs = 100;
net = train (net, P, T );
```

并对训练好的网络进行仿真：

```
[ y, pf, af, e, perf ] = sim ( net, P, {0}, { }, T )
```

仿真结果为

```
y =
    Columns 1 through 7
      [0.0707]  [-0.8369]  [-0.0105]  [1.9672]  [1.0595]  [-0.8369]  [-0.0105]
    Columns 8 through 14
      [1.0595]  [0.0707]  [0.9783]  [1.9672]  [1.0595]  [-0.8369]  [-1.8258]
    Columns 15 through 19
      [-0.0105]  [1.9672]  [1.9672]  [1.0595]  [-0.8369]
pf =
    [-1]
af =
    Empty cell array: 1-by-0
e =
    Columns 1 through 7
      [-0.0707]  [-0.1631]  [0.0105]  [0.0328]  [-0.0595]  [-0.1631]  [0.0105]
    Columns 8 through 14
      [-0.0595]  [-0.0707]  [0.0217]  [0.0328]  [-0.0595]  [-0.1631]  [-0.1742]
    Columns 15 through 19
      [0.0105]  [0.0328]  [0.0328]  [-0.0595]  [0.8369]
perf =
    0.0442
```

其中，perf为网络输出与目标矢量的均方误差。

参见：adapt、sim、init 和 revert 函数。

3.2.3 神经网络的初始化函数

网络的初始化函数如表 3-4 所示。

表 3-4 网络初始化函数

函 数 名 称	功 能
Initlay	对神经网络逐层进行初始化

initlay

功能：对神经网络逐层进行初始化。



格式：

① net = initlay (net)

② info = initlay (code)

说明：

该函数根据网络各层的初始化函数 net.layers{i}.initFcn 逐层对网络进行初始化。

网络的调用形式①返回初始化后的网络 net。调用形式②返回函数的有关信息，code 字符串可以是'pnames'或'pdefaults'，其中前者表示初始化函数各参数的名称，后者表示初始化函数各参数的取值。initlay 函数没有参数。

网络属性设置：

newp、newlin、newff、newcf 等函数建立的网络都采用 initlay 作为初始化函数，如果要使自定义网络利用 initlay 函数进行初始化，网络可作如下设置：

(1) 将网络初始化函数 net.initFcn 设置为'initlay'。由于 initlay 函数没有参数，这时网络初始化参数自动设置为[ ]；

(2) 设置网络每层的初始化函数 net.layers{i}.initFcn。

参见：initwb、initnw 和 init 函数。

3.2.4　层初始化函数

网络的层初始化函数如表 3-5 所示。

表 3-5　层初始化函数

函 数 名 称	功　　能
initnw	采用 Nguyen-Widrow 方法对网络层进行初始化
initwb	根据已设定的权值和阈值初始化函数对网络层进行初始化

1. initnw

功能：采用 Nguyen-Widrow 方法对网络层进行初始化。

格式：

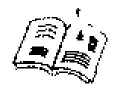
net = initnw (net, i)

说明：

该函数利用 Nguyen-Widrow 方法对网络 net 中的第 i 层进行初始化，采用该方法初始化后，每个神经元的激活区域将均匀地分布在输入空间中，从而可以避免神经元的浪费，同时提高了训练效率。只有当第 i 层神经元具有阈值，加权函数为 dotprod，输入函数为 netsum 时，才能使用 initnw 函数进行初始化。

网络属性设置：

newff、newcf 等函数建立的网络采用 initnw 作为层初始化函数，如果要使自定义网络的第 i 层利用 initnw 函数进行层初始化，则网络可作如下设置：



(1) 将网络初始化函数 net.initFcn 设置为'initlay';

(2) 将层初始化函数 net.layers{i}.initFcn 设置为'initnw'。

参见：initwb、initlay 和 init 函数。

2. initwb

功能：根据已设定的权值和阈值初始化函数对网络层进行初始化。

格式：

net = initwb (net, i)

说明：

该函数根据网络 net 中第 *i* 层神经元权值和阈值的初始化函数对该层进行初始化。

网络属性设置：

newp、newlin 等函数建立的网络采用 initnw 作为层初始化函数，如果要使自定义网络的第 *i* 层利用 initwb 函数进行层初始化，则网络可作如下设置：

(1) 将网络初始化函数 net.initFcn 设置为'initlay';

(2) 将层初始化函数 net.layers{i}.initFcn 设置为'initwb';

(3) 设置输入层权值、网络层权值和阈值的初始化函数 net.inputWeights{i,j}.initFcn、net.layerWeights{i,j}.initFcn 和 net.biases{i}.initFcn。

参见：initnw、initlay 和 init 函数。

3.2.5 权值和阈值初始化函数

权值和阈值初始化函数如表 3-6 所示。

网络属性设置：

如果要设置网络第 *i* 层的权值和阈值初始化函数，可按如下步骤进行：

(1) 将网络初始化函数 net.initFcn 设置为'initlay';

(2) 将该层的初始化函数 net.layers{i}.initFcn 设置为'initwb';

(3) 将该层的输入权值、网络权值和阈值的初始化函数 net.inputWeights{i,j}.initFcn、net.layerWeights{i,j}.initFcn 和 net.biases{i}.initFcn 分别设置为期望的初始化函数。

表 3-6 权值和阈值初始化函数

函数名称	功能
initcon	对阈值进行公平初始化
initzero	把权值和阈值初始化为零
midpoint	把权值初始化为输入矢量值域的中心
randnc	对权值进行列归一化初始化
randnr	对权值进行行归一化初始化
rands	对权值和阈值进行随机初始化

1. initcon

功能： 对阈值进行公平初始化。

格式：

B = initcon (S, PR)

说明：

当网络阈值学习函数为 learncon 时，需要采用该函数对阈值进行初始化，由于该函数返回的各神经元阈值相等，因此称为公平初始化函数。该函数一般不用来对权值进行初始化。

函数的输入 S 为神经元个数；PR 为表示输入矢量取值范围的矩阵[Pmin Pmax]，缺省值为[1 1]；函数返回初始阈值矢量 B。

例 3.20 利用 initcon 函数对一个两神经元网络层的阈值进行初始化。

```
b = initcon ( 2 )
b =
    5.4366
    5.4366
```

参见： initwb、initlay、init 和 learncon 函数。

2. initzero

功能： 把权值和阈值初始化为零。

格式：

① W = initzero (S, PR)

② B = initzero (S, [1 1])

说明：

该函数把阈值和权值都初始化为零。

函数的输入 S 为神经元个数；调用形式①中的 PR 为表示输入矢量取值范围的矩阵[Pmin Pmax]。采用调用形式①时，函数返回权值矩阵 W；采用调用形式②时，函数返回阈值矢量 B。

例 3.21 利用 initzero 函数对一个具有两神经元二输入的网络层进行初始化：

```
W = initzero ( 2, [0  1; -1  1])
b = initzero ( 2, [0  1] )
W =
    0        0
    0        0
b =
    0
    0
```

参见： initwb、initlay 和 init 函数。

3. midpoint

功能： 把权值初始化为输入矢量值域的中心。



格式：

W = midpoint (S，PR)

说明：

该函数把权值初始化为输入矢量值域的中心。

函数的输入 S 为神经元个数；PR 是表示输入矢量取值范围的矩阵[Pmin　Pmax]。函数返回权值矩阵 W，矩阵中的每一行设置为(Pmin+Pmax)/2。

例 3.22 利用 midpoint 函数对一个具有两神经元二输入的网络层权值进行初始化。

```
W = midpoint ( 2, [0  1; -1  1] )
W =
      0.5000        0
      0.5000        0
```

参见： initwb、initlay 和 init 函数。

4. randnc

功能： 对权值进行列归一化初始化。

格式：

① W = randnc (S, PR)

② W = randnc (S, R)

说明：

输入 S 为神经元个数；函数调用形式①中的 PR 是表示输入矢量取值范围的矩阵[Pmin Pmax]。调用形式②中，R 为输入矢量的维数。函数返回一个随机权值矩阵 W，矩阵中的每一列都经过了归一化。

例 3.23 产生一个三行三列的列归一化随机权值矩阵。

```
W = randnc ( 3, 3 )
W =
     0.8412   -0.0298   -0.0750
    -0.5025    0.8305   -0.8294
     0.1997    0.5563    0.5536
```

参见： randnr 函数。

5. randnr

功能： 对权值进行行归一化初始化。

格式：

① W = randnr (S, PR)

② W = randnr (S, R)

说明：

输入 S 为神经元个数；调用形式①中的 PR 是表示输入矢量取值范围的矩阵[Pmin Pmax]，调用形式②中的 R 为输入矢量的维数。函数返回一个随机权值矩阵 W，矩阵中的每一行都经过了归一化。

例 3.24　产生一个三行三列的行归一化随机权值矩阵。

W = randnr (3, 3)

W =

−0.1269	0.9680	−0.2164
0.2265	0.4674	0.8545
0.4840	−0.5367	0.6912

参见：randnc 函数。

6. Rands

功能：对权值和阈值进行随机初始化。

格式：

① W = rands (S, PR)

② M = rands (S, R)

③ B = rands (S)

说明：

输入 S 为神经元个数；调用形式①中的 PR 为表示输入矢量取值范围的矩阵[Pmin Pmax]，调用形式②中的 R 为输入矢量的维数。调用形式①和②都可以返回随机权值矩阵，调用形式③返回阈值矢量。

例 3.25　产生一个三行三列的随机权值矩阵。

W = rands (3, 3)

W =

−0.6131	0.0833	−0.2433
0.3644	−0.6983	0.7200
−0.3945	0.3958	0.7073

参见：randnr、randnc、initlay、initwb 和 init 函数。

3.2.6　训练和自适应调整函数

网络的训练和自适应调整函数如表 3-7 所示。

表 3-7　训练和自适应调整函数

函数名称	功能
trainb	根据已设定的权值和阈值学习函数对网络进行批量训练
trainc	根据已设定的权值和阈值学习函数对网络进行循环训练
trainr	根据已设定的权值和阈值学习函数对网络进行随机训练
trains	根据已设定的权值和阈值学习函数对网络进行顺序训练
trainbr	采用贝叶斯正则化算法对网络进行训练
trainbfg	采用 BFGS 准牛顿反向传播算法对网络进行训练
traincgb	采用 Powell-Beale 共轭梯度反向传播算法对网络进行训练



续表

traincgf	采用 Fletcher-Powell 共轭梯度反向传播算法对网络进行训练
traincgp	采用 Polak-Ribiere 共轭梯度反向传播算法对网络进行训练
trainscg	采用尺度化共轭梯度反向传播算法对网络进行训练
traingd	采用梯度下降反向传播算法对网络进行训练
traingda	采用自适应学习速率梯度下降反向传播算法对网络进行训练
traingdm	采用动量梯度下降反向传播算法对网络进行训练
traingdx	采用自适应学习速率动量梯度下降反向传播算法对网络进行训练
trainlm	采用 Levenberg-Marquardt 反向传播算法对网络进行训练
trainoss	采用一步正割反向传播算法对网络进行训练
trainrp	采用弹性反向传播算法对网络进行训练

表 3-7 中的函数在程序中不被直接调用，当运行 adapt 和 train 函数对网络进行训练时，adapt 和 train 函数将根据网络的自适应调整函数属性 net.adaptFcn 和训练函数属性 net.trainFcn 调用相应的训练函数。

1. trainb

功能：根据已设定的权值和阈值学习函数对网络进行批量训练。

格式：

① [net, TR, Ac, El] = trainb (net, Pd, Tl, Ai, Q, TS, VV, TV)

② info = trainb (code)

说明：

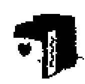
trainb 函数实现对网络的批量式训练。所谓批量训练，是指网络在接收全部样本矢量数据后，才对网络的权值和阈值进行调整。当训练函数属性 net.trainFcn 为'trainb'时，网络具有以下训练参数：

- net.trainParam.epochs：最大训练次数，网络每接收一轮输入数据并进行调整称为一次训练，该参数的缺省值为 100；
- net.trainParam.goal：网络性能目标，缺省值为 0；
- net.trainParam.max_fail：最大验证失败次数，缺省值为 5；
- net.trainParam.show：两次显示之间的训练次数，缺省值为 25；
- net.trainParam.time：最长训练时间（以秒计），缺省值为 inf。

在网络训练过程中，只要满足下列四个条件之一，网络训练便会停止：

- 达到最大训练次数 net.trainParam.epochs；
- 网络误差性能降低到目标值 net.trainParam.goal ；
- 训练时间达到最大值 net.trainParam.time；
- 连续验证失败次数达到最大次数 net.trainParam.max_fail。所谓验证失败，是指调整后的网络对验证输入矢量的输出误差没有降低。

在函数的调用形式①中，Pd 为延迟输入；Tl 为每层的目标矢量；Ai 为初始输入条件；Q 为批量输入数据的个数；TS 为时间步数；VV 为验证矢量；TV 为测试矢量。函数



返回已训练好的神经网络对象 net、训练记录 TR、最后一次仿真时各层的输出 Ac 和误差 E1，其中，训练记录 TR 包括训练次数 TR.epoch、训练性能 TR.perf、验证性能 TR.vperf 和测试性能 TR.tperf。

函数的调用形式②将返回训练函数的有关信息，code 字符串的取值为：

- 'pnames'：返回训练参数的名称；
- 'pdefaults'：返回训练参数的缺省值。

网络属性设置：

newlin 函数建立的网络采用 trainb 作为训练函数，如果要使自定义网络利用 trainb 函数进行训练，可作如下设置：

(1) 将网络训练函数 net.trainFcn 设置为'trainb'；

(2) 设置网络训练函数的参数 net.trainParam；

(3) 设置各输入权值、网络权值和阈值的学习函数 net.inputWeights{i,j}.learnFcn、net.layerWeights{i,j}.learnFcn 和 net.biases{i}.learnFcn；

(4) 设置各权值和阈值学习函数的参数 net.inputWeights{i,j}.learnParam、net.layerWeights{i,j}.learnParam 和 net.biases{i}.learnParam。

参见： newp、newlin 和 train 函数。

2. trainc

功能： 根据已设定的权值和阈值学习函数对网络进行循环训练。

格式：

① [net, TR, Ac, El] = trainc (net, Pd, Tl, Ai, Q, TS, VV, TV)

② info = trainc (code)

说明：

trainc 函数对网络进行循环训练。所谓循环训练，是指网络每接收一个输入数据，都要对权值和阈值进行调整，而且输入数据是以循环队列的形式进行排列的。网络训练函数属性 net.rainFcn 为'trainc'时，网络的训练参数包括：

- net.trainParam.epochs：最大训练次数，网络每接收一轮输入数据并进行权值和阈值调整称为一次训练，该参数缺省值为 100；
- net.trainParam.goal：网络性能目标，缺省值为 0；
- net.trainParam.show：两次显示之间的训练次数，缺省值为 25；
- net.trainParam.time：最长训练时间（以秒计），缺省值为 inf。

在网络训练过程中，只要满足下列三个条件之一，网络训练便会停止：

- 达到最大训练次数 net.trainParam.epochs；
- 网络误差性能降低到目标值 net.trainParam.goal；
- 训练时间达到最大值 net.trainParam.time。

trainc 函数在调用时各输入量和返回量的含义参见 trainb 函数。函数输出的训练记录 TR 由训练次数 TR.epoch 和训练性能 TR.perf 组成。trainc 函数在训练网络时不进行验证和测试，因此调用时可省去验证矢量 VV 和测试矢量 TV。



网络属性设置：

newp 函数建立的网络采用 trainc 作为训练函数，如果要使自定义网络利用 trainc 函数进行训练，可作如下设置：

(1) 将网络训练函数 net.trainFcn 设置为'trainc'；

(2) 设置网络训练函数的参数 net.trainParam；

(3) 设置各输入权值、网络权值和阈值的学习函数 net.inputWeights{i,j}.learnFcn、net.layerWeights{i,j}.learnFcn 和 net.biases{i}.learnFcn；

(4) 设置各权值和阈值学习函数的参数 net.inputWeights{i,j}.learnParam、net.layerWeights{i,j}.learnParam 和 net.biases{i}.learnParam。

参见： newp、newlin 和 train 函数。

3. trainr

功能： 根据已设定的权值和阈值学习函数对网络进行随机训练。

格式：

① [net, TR, Ac, El] = trainr (net, Pd, Tl, Ai, Q, TS, VV, TV)

② info = trainr (code)

说明：

trainr 函数对网络进行随机训练。所谓随机训练，是指网络每接收一个输入数据，都要对权值和阈值进行调整，而且输入数据是以随机顺序排列的。当网络训练函数的属性 net.rainFcn 为'trainr'时，网络的训练参数和训练停止条件与 trainc 函数相同。

trainr 函数在调用时各输入量和返回量的含义参见 trainb 函数。函数输出的训练记录 TR 中包括训练次数 TR.epoch 和训练性能 TR.perf。trainr 函数在训练网络时不进行验证和测试，因此调用时可省去验证矢量 VV 和测试矢量 TV。

网络属性设置：

newc 和 newsom 函数建立的网络都采用 trainr 作为训练函数，如果要使自定义网络利用 trainr 函数进行训练，可作如下设置：

(1) 将网络的训练函数 net.trainFcn 设置为'trainr'；

(2) 设置网络训练函数的参数 net.trainParam；

(3) 设置各输入权值、网络权值和阈值的学习函数 net.inputWeights{i,j}.learnFcn、net.layerWeights{i,j}.learnFcn 和 net.biases{i}.learnFcn；

(4) 设置各权值和阈值学习函数的参数 net.inputWeights{i,j}.learnParam、net.layerWeights{i,j}.learnParam 和 net.biases{i}.learnFaram。

参见： newc、newsom、newp、newlin 和 train 函数。

4. trains

功能： 根据已设定的权值和阈值学习函数对网络进行顺序训练。

格式：

① [net, TR, Ac, El] = trains (net, Pd, Tl, Ai, Q, TS, VV, TV)

② info = trains (code)



说明：

trains 函数依据已设定的权值和阈值学习函数对网络进行顺序训练。所谓顺序训练，是指输入数据按排列顺序依次进入网络，网络每接收一个输入，就对权值和阈值进行一次调整。该函数多设置为网络的自适应调整函数。当网络的自适应调整函数属性 net.adaptFcn 为'trains'时，网络的自适应调整参数为：

net.adaptParam.passes：自适应调整次数。网络接收一轮输入数据后并对权值和阈值进行更新称为一次调整，该参数缺省值为 1。

当网络训练函数属性 net.trainFcn 为'trains'时，网络的训练参数为

net.trainParam.passes：训练次数，缺省值为 1。

trains 函数在调用时各输入量和返回量的含义参见 trainb 函数。函数输出的训练记录 TR 由网络训练的时间步数 TR.timesteps 和训练性能 TR.perf 两项组成。

网络属性设置：

newp 和 newlin 函数建立的网络都采用 trains 函数作为自适应调整函数，如果要使自定义网络利用 trains 函数进行调整，可作如下设置：

(1) 将网络训练函数 net.adaptFcn 设置为'trains'；

(2) 设置网络训练函数的参数 net.adaptParam；

(3) 设置各输入权值、网络权值和阈值的学习函数 net.inputWeights{i,j}.learnFcn、net.layerWeights{i,j}.learnFcn 和 net.biases{i}.learnFcn；

(4) 设置各权值和阈值学习函数的参数 net.inputWeights{i,j}.learnParam、net.layerWeights{i,j}.learnParam 和 net.biases{i}.learnParam。

参见：newp、newlin、trainb、trainc、trainr 和 train 函数。

5. trainbr

功能：采用贝叶斯正则化算法对网络进行训练。

格式：

① [net, TR, Ac, El] = trainbr (net, Pd, Tl, Ai, Q, TS, VV, TV)

② info = trainbr (code)

说明：

trainbr 函数依据 Levenberg-Marquardt 优化理论对网络的权值和阈值进行调整（参见 trainlm 函数）。所谓 Bayesian 正则化，是指为了提高网络的推广能力，训练过程中要建立一个由各层输出误差、权值和阈值构成的特殊性能参数（该参数的定义类似于规则化均方误差函数 msereg），通过调整权值和阈值，使该参数最小化。trainbr 函数的优点在于可以提高网络推广能力，而且不会出现“过度训练”的情况。当网络输入和目标矢量的取值为[-1, 1]时，trainbr 函数可以达到最好的工作效果，因此在利用 trainbr 函数对网络进行训练之前通常应预先对样本数据作归一化处理。

当网络的训练函数 net.trainFcn 为'trainbr'时，网络的训练参数为：

- net.trainParam.epochs：训练次数，缺省值为 100；
- net.trainParam.goal：网络性能目标，缺省值为 0；
- net.trainParam.mu：Marquardt 调整参数，缺省值为 0.005；



- net.trainParam.mu_dec：mu 的下降因子，缺省值为 0.1；
- net.trainParam.mu_inc：mu 的上升因子，缺省值为 10；
- net.trainParam.mu_max：mu 的最大值，缺省值为 1e−10；
- net.trainParam.max_fail：最大验证失败次数，缺省值为 5；
- net.trainParam.mem_reduc：该参数用于权衡计算雅可比矩阵时占用的内存空间和计算速度，缺省值为 1。当该因子为 1 时，计算时占用的内存最大，但计算速度最快，当该因子增大时，计算时占用的内存减小，但计算速度也随之减慢；
- net.trainParam.min_grad：性能函数的最小梯度，缺省值为 1e−10；
- net.trainParam.show：两次显示之间的训练次数，缺省值为 25；
- net.trainParam.time：最长训练时间（以秒计），缺省值为 inf。

在网络训练过程中，只要满足下列六个条件之一，网络训练便会停止：

- 达到最大训练次数 net.trainParam.epochs；
- 网络误差性能降低到目标值 net.trainParam.goal；
- 训练时间达到最大值 net.trainParam.time；
- 性能函数梯度降低到最小梯度 net.trainParam.min_grad；
- 调整参数 mu 达到最大值 net.trainParam.mu_max；
- 连续验证失败次数达到最大次数 net.trainParam.max_fail。

trainbr 函数在调用时各输入量和返回量参见 trainb 函数。函数输出的训练记录 TR 由训练次数 TR.epoch、网络训练性能 TR.perf、验证性能 TR.vperf、测试性能 TR.tperf 和自适应调整参数 TR.mu 组成。

网络属性设置：

newff、newcf 和 newelm 函数建立的网络都可以采用 trainbr 函数作为训练函数。只要网络的加权函数、输入函数和传递函数是可微的，那么该网络就可以利用 trainbr 函数进行训练，如果要使网络利用 trainbr 函数进行训练，可作如下设置：

(1) 将网络训练函数 net. trainFcn 设置为'trainbr'；

(2) 设置网络训练函数的参数 net.trainParam。

参见： newff、newcf、newelm、traingdm、traingda、traingdx、trainlm、trainrp、traincgf、traincgb、trainscg、traincgp 和 trainoss 函数。

6. trainbfg

功能： 采用 BFGS 准牛顿反向传播算法对网络进行训练。

格式：

① [net, TR, Ac, El] = trainbfg (net, Pd, Tl, Ai, Q, TS, VV, TV)

② info = trainbfg (code)

说明：

trainbfg 函数依据 BFGS 准牛顿反向传播算法对网络的权值和阈值进行调整，该算法收敛较快，但占用的内存比共轭梯度算法要大，因此适用于小规模网络。

trainbfg 函数以及其他一些采用准牛顿算法和共轭梯度算法的训练函数都要使用线搜索函数，因此当网络的训练函数 net.trainFcn 设为此类函数时，网络的训练参数将由常规参数



和线搜索参数组成，其中常规参数包括：

- net.trainParam.epochs：训练次数，缺省值为 100；
- net.trainParam.goal：网络性能目标，缺省值为 0；
- net.trainParam.max_fail：最大验证失败次数，缺省值为 5；
- net.trainParam.min_grad：性能函数的最小梯度，缺省值为 1e-6；
- net.trainParam.show：两次显示之间的训练次数，缺省值为 25；
- net.trainParam.time：最长训练时间（以秒计），缺省值为 inf。

线搜索参数包括：

- net.trainParam.searchFcn：训练时采用的线搜索方法，缺省值为'srchcha'；
- net.trainParam.scal_tol：确定线搜索误差的尺度因子，缺省值为 20，初始步长 delta 除以该参数后得到线搜索结果的误差；
- net.trainParam.alpha：确定性能函数下降是否足够大的尺度因子，缺省值为 0.001；
- net.trainParam.beta：确定步长是否足够大的尺度因子，缺省值为 0.1；
- net.trainParam.delta：区间定位时的初始步长 ，缺省值为 0.01 ；
- net.trainParam.gama：避免性能函数下降过小的参数，缺省值为 0.1 ；
- net.trainParam.low_lim：步长变化下界，缺省值为 0.1；
- net.trainParam.up_lim：步长变化上界，缺省值为 0.5；
- net.trainParam.maxstep：最大步长，缺省值为 100；
- net.trainParam.minstep：最小步长，缺省值为 1.0 e-6；
- net.trainParam.bmax：最大步长（与 maxstep 在不同搜索算法中分别使用），缺省值为 26。

在网络训练过程中，只要满足下列五个条件之一，网络训练便会停止：

- 达到最大训练次数 net.trainParam.epochs；
- 网络误差性能降低到目标值 net.trainParam.goal；
- 训练时间达到最大值 net.trainParam.time；
- 性能函数梯度降低到最小值 net.trainParam.min_grad；
- 连续验证失败次数超过最大次数 net.trainParam.max_fail。

trainbfg 函数在调用时各输入量和返回量的含义参见 trainb 函数。

网络属性设置：

newff、newcf 和 newelm 函数建立的网络都可以采用 trainbfg 函数作为训练函数。只要网络的加权函数、输入函数和传递函数是可微的，那么该网络就可以利用 trainbfg 函数进行训练，如果要使网络利用该函数进行训练，可作如下设置：

(1) 将网络训练函数 net. trainFcn 设置为'trainbfg'；

(2) 设置网络训练函数的参数 net.trainParam。

参见： newff、newcf、newelm、traingdm、traingda、traingdx、trainlm、trainrp、traincgf、traincgb、trainscg、traincgp 和 trainoss 函数。



7. traincgb

功能： 采用 Powell-Beale 共轭梯度反向传播算法对网络进行训练。

格式：

(1) [net, TR, Ac, El] = traincgb (net, Pd, Tl, Ai, Q, TS, VV, TV)

(2) info = traincgb (code)

说明：

traincgb、traincgf、traincgp 和 trainscg 函数都是依据共轭梯度反向传播算法对网络的权值和阈值进行调整的。在共轭梯度算法中，网络参数不是沿性能曲面的快速下降方向（负梯度方向）进行调整的，而是沿先前调整方向的共轭方向进行调整的，这种方法可以加快网络的收敛速度。共轭梯度算法速度快，占用内存少，因此适用于大规模的神经网络。在共轭梯度算法中，调整方向要周期性地重新设置为性能曲面上当前点的负梯度，traincgb 函数采用 Powell-Beale 方法确定网络训练过程中何时进行这种设置。

当网络训练函数为 traincgb 时，网络的训练参数和停止条件与 trainbfg 函数的相同。traincgb 函数在调用时各输入量和返回量的含义参见函数 trainb。

网络属性设置：

newff、newcf 和 newelm 函数建立的网络都可以采用 traincgb 函数作为训练函数。只要网络的加权函数、输入函数和传递函数是可微的，那么该网络就可以利用 traincgb 函数进行训练，如果要使网络利用该函数进行训练，可作如下设置：

(1) 将网络训练函数 net. trainFcn 设置为'traincgb'；

(2) 设置网络训练函数的参数 net.trainParam。

参见： newff、newcf、newelm、traingdm、traingda、traingdx、trainlm、trainrp、traincgf、trainscg、traincgp、trainbfg 和 trainoss 函数。

8. traincgf

功能： 采用 Fletcher-Powell 共轭梯度反向传播算法对网络进行训练。

格式：

① [net, TR, Ac, El] = traincgf (net, Pd, Tl, Ai, Q, TS, VV, TV)

② info = traincgf (code)

说明：

traincgf 函数是共轭梯度算法的一种变形。当网络训练函数为 traincgf 时，网络的训练参数和停止条件与 trainbfg 函数相同。

traincgf 函数在调用时各输入量和返回量的含义参见函数 trainb。

网络属性设置：

newff、newcf 和 newelm 函数建立的网络都可以采用 traincgf 函数作为训练函数。只要网络的加权函数、输入函数和传递函数是可微的，那么该网络就可以利用 traincgf 函数进行训练，如果要使网络利用该函数进行训练，可作如下设置：

(1) 将网络训练函数 net. trainFcn 设置为'traincgf'；

(2) 设置网络训练函数的参数 net.trainParam。



参见： newff、newcf、newelm、traingdm、traingda、traingdx、trainlm、trainrp、traincgb、trainscg、traincgp、trainbfg 和 trainoss 函数。

9. traincgp

功能： 采用 Polak-Ribiere 共轭梯度反向传播算法对网络进行训练。

格式：

① [net, TR, Ac, El] = traincgp (net, Pd, Tl, Ai, Q, TS, VV, TV)

② info = traincgp (code)

说明：

traincgp 函数是共轭梯度算法的一种变形。当网络训练函数为 traincgp 时，网络的训练参数和停止条件与 trainbfg 函数相同。

traincgp 函数在调用时各输入量和返回量的含义参见函数 trainb。

网络属性设置：

newff、newcf 和 newelm 函数建立的网络都可以采用 traincgp 函数作为训练函数。只要网络的加权函数、输入函数和传递函数是可微的，那么该网络就可以利用 traincgp 函数进行训练，如果要使网络利用该函数进行训练，可作如下设置：

(1) 将网络训练函数 net.trainFcn 设置为'traincgp'；

(2) 设置网络训练函数的参数 net.trainParam。

参见： newff、newcf、newelm、traingdm、traingda、traingdx、trainlm、trainrp、traincgb、trainscg、traincgf、trainbfg 和 trainoss 函数。

10. trainscg

功能： 采用尺度化共轭梯度反向传播算法对网络进行训练。

格式：

① [net, TR, Ac, El] = trainscg (net, Pd, Tl, Ai, Q, TS, VV, TV)

② info = trainscg (code)

说明：

trainscg 函数是共轭梯度算法的一种变形，该算法结合了 Levenberg-Marquardt 算法中的模型置信区间方法和共轭梯度算法，避免了耗时巨大的线搜索过程，从而提高了网络的训练速度。当网络训练函数为 trainscg 时，网络的训练参数为：

- net.trainParam.epochs：训练次数，缺省值为 100；
- net.trainParam.goal：网络性能目标，缺省值为 0；
- net.trainParam.max_fail：最大验证失败次数，缺省值为 5；
- net.trainParam.min_grad：性能函数的最小梯度，缺省值为 1e−6；
- net.trainParam.show：两次显示之间的训练次数，缺省值为 25；
- net.trainParam.time：最长训练时间（以秒计），缺省值为 inf；
- net.trainParam.sigma：用于二阶导数逼近的权值变化参数，缺省值为 5.0 e−5；
- net.trainParam.lambda：Hessian 矩阵不确定性的调整参数，缺省值为 5.0 e−7。



此时网络训练的停止条件与函数 trainbfg 的相同。

trainscg 函数在调用时各输入量和返回量的含义参见函数 trainb。

网络属性设置：

newff、newcf 和 newelm 函数建立的网络都可以采用 trainscg 函数作为训练函数。只要网络的加权函数、输入函数和传递函数是可微的，那么该网络就可以利用 trainscg 函数进行训练。如果要使网络利用 trainscg 函数进行训练，可作如下设置：

(1) 将网络训练函数 net. trainFcn 设置为'trainscg'；

(2) 设置网络训练函数的参数 net.trainParam。

参见： newff、newcf、newelm、traingdm、traingda、traingdx、trainlm、trainrp、traincgb、traincgf、traincgp、trainbfg 和 trainoss 函数

11. traingd

功能： 采用梯度下降反向传播算法对网络进行训练。

格式：

① [net, TR, Ac, El] = traingd (net, Pd, Tl, Ai, Q, TS, VV, TV)

② info = traingd (code)

说明：

traingd 函数实现了最基本的最速梯度下降算法，即基本 BP 算法，其他梯度下降算法都是该算法的变形。当网络训练函数为 traingd 时，网络的训练参数为：

- net.trainParam.epochs：训练次数，缺省值为 100；
- net.trainParam.goal：网络性能目标，缺省值为 0；
- net.trainParam.lr：学习速率，缺省值为 0.01；
- net.trainParam.max_fail：最大验证失败次数，缺省值为 5；
- net.trainParam.min_grad：性能函数的最小梯度，缺省值为 1e-10；
- net.trainParam.show：两次显示之间的训练次数，缺省值为 25；
- net.trainParam.time：最长训练时间（以秒计），缺省值为 inf。

此时网络训练的停止条件与函数 trainbfg 的相同。

traingd 函数在调用时各输入量和返回量的含义参见函数 trainb。

网络属性设置：

newff 、newcf 和 newelm 函数建立的网络都可以采用 traingd 函数作为训练函数。只要网络的加权函数、输入函数和传递函数是可微的，那么该网络就可以利用 traingd 函数进行训练。如果要使自定义网络利用 traingd 函数进行训练，可作如下设置：

(1) 将网络训练函数 net. trainFcn 设置为'traingd'；

(2) 设置网络训练函数的参数 net.trainParam。

参见： newff、newcf、newelm、traingdm、traingda、traingdx 和 trainlm 函数。

12. traingda

功能： 采用自适应学习速率梯度下降反向传播算法对网络进行训练。



格式：

① [net, TR, Ac, El] = traingda (net, Pd, Tl, Ai, Q, TS, VV, TV)

② info = traingda (code)

说明：

traingda 函数是最速梯度下降算法的一种变形，由于网络的性能对学习速率十分敏感，因此该函数在网络训练过程中对学习速率进行自适应调整，从而提高了网络的训练效率。当网络训练函数为 traingda 时，网络的训练参数为：

- net.trainParam.epochs：训练次数，缺省值为 100；
- net.trainParam.goal：网络性能目标，缺省值为 0；
- net.trainParam.lr：学习速率，缺省值为 0.01；
- net.trainParam.lr_inc：学习速率增长比例因子，缺省值为 1.05；
- net.trainParam.lr_dec：学习速率下降比例因子，缺省值为 0.7；
- net.trainParam.max_fail：最大验证失败次数，缺省值为 5；
- net.trainParam.max_perf_inc：性能参数最大增长值，缺省值为 1.04；
- net.trainParam.min_grad：性能函数的最小梯度，缺省值为 1e−10；
- net.trainParam.show：两次显示之间的训练次数，缺省值为 25；
- net.trainParam.time：最长训练时间（以秒计），缺省值为 inf。

此时网络训练的停止条件和函数 trainbfg 的相同。

traingda 函数在调用时各输入量和返回量的含义参见函数 trainb。函数输出的训练记录由训练次数 TR.epoch、训练性能 TR.perf、验证性能 TR.vperf、测试性能 TR.tperf 和自适应学习速率 TR.lr 组成。

网络属性设置：

newff、newcf 和 newelm 函数建立的网络都可以采用 traingda 函数作为训练函数。只要网络的加权函数、输入函数和传递函数都是可微的，那么该网络就可以利用 traingda 函数进行训练。如果要使网络利用 traingda 函数进行训练，可作如下设置：

(1) 将网络训练函数 net. trainFcn 设置为'traingda'；

(2) 设置网络训练函数的参数 net.trainParam。

参见： newff、newcf、newelm、traingdm、traingd、traingdx 和 trainlm 函数。

13. traingdm

功能： 采用动量梯度下降反向传播算法对网络进行训练。

格式：

① [net, TR, Ac, El] = traingdm (net, Pd, Tl, Ai, Q, TS, VV, TV)

② info = traingdm (code)

说明：

traingdm 函数是最速梯度下降算法的一种变形，该函数在对权值和阈值更新时不仅考虑当前的梯度方向，而且还考虑了前一时刻的梯度方向，从而降低了网络性能对参数调整的敏感性，有效地抑制了局部极小，这种调整方式是通过设置动量因子来实现的。当网络训练函数为 traingdm 时，网络的训练参数为：



- net.trainParam.epochs：训练次数，缺省值为 100；
- net.trainParam.goal：网络性能目标，缺省值为 0；
- net.trainParam.lr：学习速率，缺省值为 0.01；
- net.trainParam.max_fail：最大验证失败次数，缺省值为 5；
- net.trainParam.mc：动量常数，缺省值为 0.9；
- net.trainParam.min_grad：性能函数的最小梯度，缺省值为 1e-10；
- net.trainParam.show：两次显示之间的训练次数，缺省值为 25；
- net.trainParam.time：最长训练时间（以秒计），缺省值为 inf。

此时网络训练的停止条件与函数 trainbfg 的相同。

traingdm 函数在调用时各输入量和返回量的含义参见函数 trainb。

网络属性设置：

newff、newcf 和 newelm 函数建立的网络都可以采用 traingdm 函数作为训练函数。只要网络的加权函数、输入函数和传递函数是可微的，那么该网络就可以利用 traingdm 函数进行训练。如果要使网络利用 traingdm 函数进行训练，可作如下设置：

(1) 将网络训练函数 net. trainFcn 设置为'traingdm'；

(2) 设置网络训练函数的参数 net.trainParam。

参见： newff、newcf、newelm、traingd、traingda、traingdx 和 trainlm 函数。

14. traingdx

功能： 采用自适应学习速率动量梯度下降反向传播算法对网络进行训练。

格式：

① [net, TR, Ac, El] = traingdx (net, Pd, Tl, Ai, Q, TS, VV, TV)

② info = traingdx (code)

说明：

traingdx 函数结合了动量梯度下降算法和自适应学习速率梯度下降算法，从而使网络的训练速度和稳定性有了进一步提高。当网络训练函数为 traingdx 时，网络的训练参数为：

- net.trainParam.epochs：训练次数，缺省值为 100；
- net.trainParam.goal：网络性能目标，缺省值为 0；
- net.trainParam.lr：学习速率，缺省值为 0.01；
- net.trainParam.lr_inc：学习速率增长比例因子，缺省值为 1.05；
- net.trainParam.lr_dec：学习速率下降比例因子，缺省值为 0.7；
- net.trainParam.max_fail：最大验证失败次数，缺省值为 5；
- net.trainParam.max_perf_inc：性能参数最大增长值，缺省值为 1.04；
- net.trainParam.mc：动量常数，缺省值为 0.9；
- net.trainParam.min_grad：性能函数的最小梯度，缺省值为 1e-10；
- net.trainParam.show：两次显示之间的训练次数，缺省值为 25；
- net.trainParam.time：最长训练时间（以秒计），缺省值为 inf。

此时网络训练的停止条件与函数 trainbfg 的相同。



traingdx 函数在调用时各输入量和返回量的含义参见函数 trainb。函数输出的训练记录由训练次数 TR.epoch、训练性能 TR.perf、验证性能 TR.vperf、测试性能 TR.tperf 和自适应学习速率 TR.lr 组成。

网络属性设置：

newff、newcf 和 newelm 函数建立的网络都可以采用 traingdx 函数作为训练函数。只要网络的加权函数、输入函数和传递函数是可微的，那么该网络都可以利用 traingdx 函数进行训练。如果要使网络利用 traingdx 函数进行训练，可作如下设置：

(1) 将网络训练函数 net. trainFcn 设置为'traingdx'；

(2) 设置网络训练函数的参数 net.trainParam。

参见： newff、newcf、newelm、traingd、traingda、traingdm 和 trainlm 函数。

15. trainlm

功能： 采用 Levenberg-Marquardt 反向传播算法对网络进行训练。

格式：

① [net, TR, Ac, El] = trainlm (net, Pd, Tl, Ai, Q, TS, VV, TV)

② info = trainlm (code)

说明：

trainlm 函数依据 Levenberg-Marquardt 优化理论对网络的权值和阈值进行调整，当采用 trainlm 作为训练函数时，网络的训练参数与函数 trainbr 的相同，其中，训练参数 mu 决定了学习算法的性质，若 mu 较大，学习过程主要依据梯度下降法，若 mu 较小，学习过程主要依据牛顿法。该训练函数的效率优于最速下降法，但占用内存较大，为了节省内存，可以把训练参数中的 net.trainFaram.mem_reduc 设置为 2 或更大的值。网络训练停止条件也与函数 trainbr 相同。

trainlm 函数在调用时各输入量和返回量的的含义参见 trainb 函数。函数输出的训练记录 TR 由训练次数 TR.epoch、训练性能 TR.perf、验证性能 TR.vperf、测试性能 TR.tperf 和自适应调整参数 TR.mu 组成。

网络属性设置：

newff、newcf 和 newelm 函数建立的网络都可以采用 trainlm 函数作为训练函数。只要网络的加权函数、输入函数和传递函数是可微的，那么该网络就可以利用 trainlm 函数进行训练。如果要使网络利用 trainlm 函数进行训练，可作如下设置：

(1) 将网络训练函数 net. trainFcn 设置为'trainlm'；

(2) 设置网络训练函数的参数 net.trainParam。

参见： newff、newcf、newelm、traingdm、traingda、traingdx、trainbr、trainrp、traincgf、traincgb、trainscg、traincgp 和 trainoss 函数。

16. trainoss

功能： 采用一步正割反向传播算法对网络进行训练。

格式：

① [net, TR, Ac, El] = trainoss (net, Pd, Tl, Ai, Q, TS, VV, TV)



② info = trainoss (code)

说明：

trainoss 函数依据一步正割反向传播算法对网络的权值和阈值进行调整，该函数是一种准牛顿算法。trainoss 函数占用的内存比共轭梯度算法大，但却小于 BFGS 算法（参见 trainbfg 函数），因此可以认为是两者的折衷。当网络训练函数为 trainoss 时，网络的训练参数和训练停止条件与 trainbfg 函数相同。

trainoss 函数在调用时所涉及参数的含义参见 trainb 函数。

网络属性设置：

newff、newcf 和 newelm 函数建立的网络都可以采用 trainoss 函数作为训练函数。只要网络的加权函数、输入函数和传递函数是可微的，那么该网络就可以利用 trainoss 函数进行训练。如果要使网络利用 trainoss 函数进行训练，可作如下设置：

(1) 将网络训练函数 net. trainFcn 设置为'trainoss'；

(2) 设置网络训练函数的参数 net.trainParam。

参见： newff、newcf、newelm、traingdm、traingda、traingdx、trainlm、trainrp、traincgf、traincgb、trainscg、traincgp 和 trainbfg 函数。

17. trainrp

功能： 采用弹性反向传播算法对网络进行训练。

格式：

① [net, TR, Ac, El] = trainrp (net, Pd, Tl, Ai, Q, TS, VV, TV)

② info = trainrp (code)

说明：

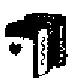
trainrp 函数依据弹性反向传播算法对网络的权值和阈值进行调整，该函数在对权值和阈值更新时只考虑梯度的符号，调整幅度则由程序设定，从而提高了网络在性能曲面平坦区域的学习效率。

当网络训练函数为 trainrp 时，网络的训练参数为：

- net.trainParam.epochs：训练次数，缺省值为 100；
- net.trainParam.goal：网络性能目标，缺省值为 0；
- net.trainParam.lr：学习速率，缺省值为 0.01；
- net.trainParam.max_fail：最大验证失败次数，缺省值为 5；
- net.trainParam.min_grad：性能函数的最小梯度，缺省值为 1e-6；
- net.trainParam.show：两次显示之间的训练次数，缺省值为 25；
- net.trainParam.time：最长训练时间（以秒计），缺省值为 inf。
- net.trainParam.delt_inc：权值变化增加量，缺省值为 1.2；
- net.trainParam.delt_dec：权值变化减少量，缺省值为 0.5；
- net.trainParam.delta0：初始权值变化量，缺省值为 0.07；
- net.trainParam.deltamax：最大权值变化量，缺省值为 50.0。

网络训练的停止条件与 trainbfg 函数的相同。

trainrp 函数在调用时各输入量和返回量的含义参见 trainb 函数。



网络属性设置：

newff、newcf 和 newelm 函数建立的网络都可以采用 trainrp 函数作为训练函数。只要网络的加权函数、输入函数和传递函数是可微的，那么该网络就可以利用 trainrp 函数进行训练。如果要使网络利用 trainrp 函数进行训练，可作如下设置：

(1) 将网络训练函数 net. trainFcn 设置为'trainrp'；

(2) 设置网络训练函数的参数 net.trainParam。

参见： newff、newcf、newelm、traingdm、traingda、traingdx、trainlm、trainoss、traincgf、traincgb、trainscg、traincgp 和 trainbfg 函数。

3.2.7 学习函数

网络权值和阈值的学习函数如表 3-8 所示。这些函数在程序中不被直接调用，当网络利用函数 trainb、trainc、trainr 和 trains 进行训练时，这些训练函数将根据已经设置好的权值和阈值学习函数的属性调用相应的学习函数。

表 3-8 学 习 函 数

函数名称	功能
learncon	公平阈值学习函数
learngd	梯度下降权值和阈值学习函数
learngdm	动量梯度下降权值和阈值学习函数
learnh	Hebb 权值学习函数
learnhd	退化 Hebb 权值学习率
learnis	星内权值学习函数
learnk	Kohonen 权值学习函数
learnlv1	LVQ1 权值学习函数
learnlv2	LVQ2 权值学习函数
learnos	星外权值学习函数
learnp	感知器权值和阈值学习函数
learnpn	归一化感知器权值和阈值学习函数
learnsom	自组织映射网络权值学习函数
learnwh	Widrow-Hoff 权值和阈值学习函数

1. learncon

功能： 公平阈值学习函数。

格式：

① [dB, LS] = learncon (B, P, Z, N, A, T, E, gW, gA, D, LP, LS)

② info = learncon (code)



说明：

learncon 函数是一种阈值学习函数，常用于调整竞争层网络中神经元的阈值，该函数通过建立一个公正参数，以保证小输出神经元的阈值大幅度增加，大输出神经元的阈值小幅度增加。该函数中阈值的调整量仅由神经元的原始阈值和输出决定。

在函数调用形式①的各输入中，B 为网络某层的阈值矢量；P 为全 1 矩阵；Z 为经过加权函数变换后的加权输入矢量；N 为加权输入经过输入函数变换后的神经元传递函数的输入矢量；A 为该层网络的输出矢量；T 为目标矢量；E 为误差矢量；gW 为网络性能对于阈值的梯度矢量；gA 为网络性能对于该层输出的梯度矢量；D 为神经元的距离矩阵；LP 为学习参数，learncon 函数的学习参数是由学习速率 LP.lr 构成的，缺省值为 0.001；LS 为学习状态，初始状态时取值为[ ]。函数返回阈值调整量 dB 和当前学习状态 LS。

函数的调用形式②返回该学习函数的有关信息。code 字符串的取值为：

- 'pnames'：返回学习参数的名称；
- 'pdefaults'：返回学习参数的缺省值；
- 'needg'：返回学习函数是否要利用梯度矢量 gW 或 gA，当用到梯度矢量时，函数返回 1。

例 3.26　假定 A 和 B 分别为两神经元网络层的输出矢量和阈值矢量：

```
A = [ 0.9501;   0.2311 ];
B = [ 0.6068;   0.4860 ];
```

设置好学习速率并利用公平学习规则对阈值进行更新：

```
LP.lr = 0.5;
dB = learncon ( B, [ ], [ ], [ ], A, [ ], [ ], [ ], [ ], [ ], LP, [ ] )
```

阈值更新量为

```
dB =
        0.3954
        0.4474
```

网络属性设置：

如果要使网络第 *i* 层神经元的阈值利用 learncon 函数进行学习，可作如下设置：

(1) 将网络训练函数 net.trainFcn 设置为'trainr'，并设置相应的训练参数；

(2) 将网络自适应调整函数 net.adaptFcn 设置为'trains'，并设置相应的自适应调整参数；

(3) 将网络第 *i* 层神经元的阈值学习函数 net.biases{i}.learnFcn 设置为'learncon'，并设置相应的学习参数。

参见： learnk、learnos、adapt 和 train 函数。

2. learngd

功能： 梯度下降权值和阈值学习函数。

格式：

① [dW, LS] = learngd (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)



② [dB, LS] = learngd (B, P, Z, N, A, T, E, gW, gA, D, LP, LS)

③ info = learngd (code)

说明：

learngd 函数采用梯度下降方法对权值和阈值进行调整，即权值和阈值的调整量 dW 为学习速率 lr 和梯度 gW 的乘积：

dW (i, j) = lr * gW (i, j)

在函数调用形式①中，W 为权值矩阵；P 为层输入矢量；Z 为层输入经过加权函数变换后的加权输入矢量；N 为加权输入经过输入函数计算后得到的神经元传递函数的输入矢量；A 为该层神经元输出矢量；T 为目标矢量；E 为误差矢量；gW 为网络性能对于权值的梯度矢量；gA 为网络性能对于该层输出的梯度矢量；D 为神经元的距离矩阵；LP 为学习参数，learngd 函数的学习参数是由学习速率 LP.lr 构成的，缺省值为 0.01；LS 为学习状态，初始值为[ ]。函数返回阈值调整量 dW 和当前学习状态 LS。函数的其余两种调用形式参见函数 learncon。

例 3.27 对一个三输入两神经元网络层随机产生梯度矢量 gW：

```
gW = rand ( 2, 3 )
gW =
        0.6154    0.9218    0.1763
        0.7919    0.7382    0.4057
```

设置好学习速率并利用梯度下降法对权值进行更新：

```
LP.lr = 0.5;
dW = learngd ( [ ], [ ], [ ], [ ], [ ], [ ], [ ], gW, [ ], [ ], LP, [ ] )
dW =
        0.3077    0.4609    0.0881
        0.3960    0.3691    0.2029
```

网络属性设置：

如果要使网络第 *i* 层神经元的权值和阈值使用 learngd 学习函数，可作如下设置：

(1) 将网络训练函数 net.trainFcn 设置为'trainb'、'trainc'或'trainr'，并设置相应的训练参数；

(2) 将网络自适应调整函数 net.adaptFcn 设置为'trains'，并设置相应的自适应调整参数；

(3) 将第 *i* 层神经元的输入权值学习函数 net.inputWeights{i,j}.learnFcn、网络权值学习函数 net.layerWeights{i,j}.learnFcn 和阈值学习函数 net.biases{i}.learnFcn 设置为 'learngd'，并设置相应的学习参数。

参见： learngdm、newff、newcf、adapt 和 train 函数。

3. learngdm

功能： 动量梯度下降权值和阈值学习函数。

格式：

① [dW, LS] = learngdm (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)



② [dB, LS] = learngdm (B, P, Z, N, A, T, E, gW, gA, D, LP, LS)

③ info = learngdm (code)

说明：

learngdm 函数采用动量梯度下降方法对权值和阈值进行调整，即权值和阈值的调整值 dW 由动量因子 mc、前一次学习时的调整量 dWprev、学习速率 lr 和梯度 gW 共同确定。

dW (i, j) = mc * dWprev (i, j) + (1−mc) * lr * gW (i, j)

函数调用时各输入量和返回量的含义参见函数 learncon 和 learngd，其中前一次学习时的权值及阈值调整量 dWprev 存储在学习状态参数 LS.dw 中。learngdm 函数的学习参数包括：

- LP.lr：学习速率，缺省值为 0.01；
- LP.mc：动量常数，缺省值为 0.9。

例 3.28 设定权值梯度矩阵 gW 和上一次的权值调整量 LS.dw 为

```
gW = [ 0.6154    0.9218    0.1763;  0.7919    0.7382    0.4057 ];
LS.dw = [ 0.8132    0.1389    0.1987;  0.0099    0.2028    0.6038 ];
```

设置学习参数并利用动量梯度下降法对权值进行更新：

```
LP.lr = 0.5;  LP.mc = 0.8;
dW = learngdm ( [ ], [ ], [ ], [ ], [ ], [ ], [ ], gW, [ ], [ ], LP, LS )
dW =
        0.7121    0.2033    0.1766
        0.0871    0.2360    0.5236
```

网络属性设置：

如果要使网络第 *i* 层神经元的权值和阈值使用 learngdm 学习函数，可作如下设置：

(1) 将网络训练函数 net.trainFcn 设置为'trainb'、'trainc'或'trainr'，并设置相应的训练参数；

(2) 将网络自适应调整函数 net.adaptFcn 设置为“trains”，并设置相应的自适应调整参数；

(3) 将网络第 *i* 层神经元的输入权值学习函数 net.inputWeights{i,j}.learnFcn、网络权值学习函数 net.layerWeights{i,j}.learnFcn 和阈值学习函数 net.biases{i}.learnFcn 设置为'learngdm'，并设置相应的学习参数。

参见： learngd、newff、newcf、adapt 和 train 函数。

4. learnh

功能： Hebb 权值学习函数。

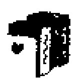
格式：

① [dW, LS] = learnh (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

② info = learnh (code)

说明：

learnh 函数采用 Hebb 联想学习规则对神经元权值进行调整，即两个神经元的网络权值



正比于它们的活动值，所以当两个神经元的输出同时为高时，这两个神经元之间将具有较大的网络权值，权值调整值 dW 由学习速率 lr、神经元输入 P 和输出 A 共同确定。

dW (i, j) = lr * A (i) * P (j)

函数调用时各输入量和返回量的含义参见函数 learngd。该函数的学习参数是由学习速率 LP.lr 构成的，缺省值为 0.01。

例 3.29 假定 P 和 A 分别为一个两神经元三输入网络层的输入和输出矢量：

```
P = [ 0.2722; 0.1988; 0.0153 ];
A= [ 0.7468;   0.4451];
```

设置学习速率并利用Hebb学习规则更新权值：

```
LP.lr = 0.5;
dW = learnh ( [ ], P, [ ], [ ], A, [ ], [ ], [ ], [ ], [ ], LP, [ ] )
```

权值更新量为

```
dW =
        0.1016      0.0742      0.0057
        0.0606      0.0442      0.0034
```

网络属性设置：

如果要使自定义网络第 *i* 层神经元的权值利用 learnh 函数进行学习，可作如下设置：

(1) 将网络训练函数 net.trainFcn 设置为'trainr'，并设置相应的训练参数；

(2) 将网络自适应调整函数 net.adaptFcn 设置为'trains'，并设置相应的自适应调整参数；

(3) 将第 *i* 层神经元的输入权值学习函数 net.inputWeights{i, j}.learnFcn 和网络权值学习函数 net.layerWeights{i,j}.learnFcn 设置为'learnh'，并设置相应的学习参数。

参见： learnhd、adapt 和 train 函数。

5. learnhd

功能： 退化 Hebb 权值学习函数。

格式：

① [dW, LS] = learnhd (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

② info = learnhd (code)

说明：

learnhd 函数是 Hebb 联想学习规则的变形，该函数中通过增加一个附加项来限制权值的规模，该附加项由衰减率 dr 和当前权值 W 确定，这时权值调整量为

dW (i, j) = lr * A (i) * P (j) − dr * W (i, j)

其中，lr为学习速率，P为神经元输入，A为输出。

函数调用时各输入量和返回量的含义参见函数 learngd。该函数的学习参数包括：

- LP.lr：学习速率，缺省值为 0.1；
- LP.dr：衰减率，缺省值为 0.01。



例 3.30 根据下列输入矢量 P、输出矢量 A 和原始权值矩阵 W，利用 learnhd 函数对神经元权值进行学习：

```
P = [ 0.2722; 0.1988; 0.0153 ];
A= [ 0.7468; 0.4451 ];
W = [ 0.6721    0.0196    0.3795;  0.8381    0.6813    0.8318 ];
```

设置学习速率、衰减率并更新权值：

```
LP.lr = 0.5;    LP.dr = 0.05;
dW = learnhd ( W, P, [ ], [ ], A, [ ], [ ], [ ], [ ], [ ], LP, [ ] )
dW =
        0.0680    0.0733    -0.0133
        0.0187    0.0102    -0.0382
```

网络属性设置：

如果要使网络第 i 层神经元的权值利用 learnhd 函数进行学习，可作如下设置：

(1) 将网络的训练函数 net.trainFcn 设置为'trainr'，并设置相应的训练参数；

(2) 将网络的自适应调整函数 net.adaptFcn 设置为'trains'，并设置相应的自适应调整参数；

(3) 将第 i 层神经元的输入权值学习函数 net.inputWeights{i,j}.learnFcn 和网络权值学习函数 net.layerWeights{i,j}.learnFcn 设置为'learnhd'，并设置相应的学习参数。

参见：learnhd、adapt 和 train 函数。

6. learnis

功能：星内权值学习函数。

格式：

① [dW, LS] = learnis (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

② info = learnis (code)

说明：

learnis 函数利用星内联想学习方法对权值进行调整。在该方法中，当神经元具有高输出时，权值将向当前输入矢量进行逼近调整，这样当网络再输入同一矢量时，该神经元将出现明显的高输出。learnis 函数中的权值调整原理为

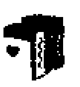
$$dW(i,j) = lr * A(i) * (P(j) - W(i,j))$$

其中，lr为学习速率，P为神经元输入，A为输出，W为当前权值矩阵。

函数调用时各输入量和返回量的含义参见函数 learngd。该函数的学习参数为学习速率 LP.lr，缺省值为 0.01。

例 3.31 根据下列输入矢量 P、输出矢量 A 和原始权值矩阵 W，利用 learnis 函数对神经元权值进行学习：

```
P = [ 0.2722; 0.1988; 0.0153 ];
A= [ 0.7468; 0.4451 ];
W = [ 0.6721    0.0196    0.3795;  0.8381    0.6813    0.8318 ];
```



设置学习速率并更新权值：

```
LP.lr = 0.5;
dW = learnis ( W, P, [ ], [ ], A, [ ], [ ], [ ], [ ], [ ], LP, [ ] )
dW =
        -0.1493     0.0669    -0.1360
        -0.1259    -0.1074    -0.1817
```

网络属性设置：

如果要使网络第 i 层神经元的权值利用 learnis 函数进行学习，可作如下设置：

(1) 将网络的训练函数 net.trainFcn 设置为'trainr'，并设置相应的训练参数；

(2) 将网络的自适应调整函数 net.adaptFcn 设置为'trains'，并设置相应的自适应调整参数；

(3) 将第 i 层神经元的输入权值学习函数 net.inputWeights{i,j}.learnFcn 和网络权值学习函数 net.layerWeights{i,j}.learnFcn 设置为'learnis'，并设置相应的学习参数。

参见： learnk、learnos、adapt 和 train 函数。

7. learnk

功能： Kohonen 权值学习函数。

格式：

① [dW, LS] = learnk (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

② info = learnk (code)

说明：

learnk 函数利用 Kohonen 规则对权值进行调整，该函数多用于竞争层网络中。Kohonen 规则可以使输入矢量存储于神经元权值中。在 learnk 函数中，当神经元输出为 0 时，权值不作调整；当输出不为 0 时，权值调整量为

dW (i, j) = lr * (P (j) - W (i, j))

其中，lr为学习速率，P为神经元输入，W为当前权值矩阵。

函数调用时各输入量和返回量的含义参见函数 learngd。该函数的学习参数为学习速率 LP.lr，缺省值为 0.01。

例 3.32 对一个三输入两神经元的网络层，设定输入矢量 P 和权值矩阵 W，并计算输出矢量 A：

```
P =  [ 0.5028;    0.7095;    0.4289 ];
W = [ 0.3046      0.1934     0.3028;
      0.1897      0.6822     0.5417 ];
A = compet ( negdist ( W, P ) )
A =
        (2,1)         1
```

然后设置学习速率，并利用 Kohonen 学习规则更新权值：

```
LP.lr = 0.5;
```



```
dW = learnk ( W, P, [ ], [ ], A, [ ], [ ], [ ], [ ], [ ], LP, [ ] )
dW =
              0          0          0
         0.1566     0.0136    -0.0564
```

网络属性设置：

如果要使自定义网络第 i 层神经元的权值利用 learnk 函数进行学习，可作如下设置：

(1) 将网络的训练函数 net.trainFcn 设置为'trainr'，并设置相应的训练参数；

(2) 将网络的自适应调整函数 net.adaptFcn 设置为'trains'，并设置相应的自适应调整参数；

(3) 将第 i 层神经元的输入权值学习函数 net.inputWeights{i,j}.learnFcn 和网络权值学习函数 net.layerWeights{i,j}.learnFcn 设置为 "learnk"，并设置相应的学习参数。

参见： learnis、learnos、adapt 和 train 函数。

8. learnlv1

功能： LVQ1 权值学习函数。

格式：

① [dW, LS] = learnlv1 (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

② info = learnlv1 (code)

说明：

learnlv1 函数利用 LVQ1 规则对权值进行调整，该函数多用于学习矢量量化网络中。在 learnlv1 函数中，当神经元输出为 0 时，权值不作调整；当输出不为 0 时，权值将根据梯度矢量 gA 的取值进行调整：

$$dW(i,j) = lr * (P(j) - W(i,j)) \quad \text{当 } gA(i) = 0$$

$$dW(i,j) = -lr * (P(j) - W(i,j)) \quad \text{当 } gA(i) \neq 0$$

其中，lr为学习速率，P为神经元输入，W为当前权值。

函数调用时各输入量和返回量的含义参见函数 learngd，其中梯度矢量 gA 用来表示该层神经元的分类结果是否正确。该函数的学习参数为学习速率 LP.lr，缺省值为 0.01。

例 3.33 根据输入矢量 P 和权值矩阵 W，利用 learnlv1 函数对神经元权值进行学习：

```
P = [ 0.5028; 0.7095; 0.4289 ];
W = [ 0.3046    0.1934    0.3028; 0.1897    0.6822    0.5417 ];
```

首先计算神经元的输出：

```
A = compet ( negdist ( W, P ) );
```

设定梯度矢量：

```
gA = [ 1; 1 ];
```

设置学习速率并更新权值：

```
LP.lr = 0.5;
dW = learnlv1 ( W, P, [ ], [ ], A, [ ], [ ], [ ], gA, [ ], LP, [ ] )
```

权值调整量为

dW =

0	0	0
-0.1566	0.0136	0.0564

网络属性设置：

如果要使网络第 i 层神经元的权值利用 learnlv1 函数进行学习，可作如下设置：

(1) 将网络的训练函数 net.trainFcn 设置为'trainr'，并设置相应的训练参数；

(2) 将网络的自适应调整函数 net.adaptFcn 设置为'trains'，并设置相应的自适应调整参数；

(3) 将网络第 i 层神经元的输入权值学习函数 net.inputWeights{i,j}.learnFcn 和网络权值学习函数 net.layerWeights{i,j}.learnFcn 设置为'learnlv1'，并设置相应的学习参数。

参见： learnlv2、adapt 和 train 函数。

9. learnlv2

功能： LVQ2 权值学习函数。

格式：

① [dW, LS] = learnlv2 (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

② info = learnlv2 (code)

说明：

learnlv2 函数利用 LVQ2 规则对权值进行调整。该函数同样用于学习矢量量化网络中，但网络只有在利用 LVQ1 规则进行学习后，才能使用 LVQ2 规则。在利用 learnlv2 函数进行学习时，如果输出最大的两个神经元分别为 i 和 j，而且输入矢量落在这两个神经元的权值矢量中间，即满足：

min (di/dj, dj/di) > (1-window) / (1+window)

其中，di 和 dj 分别为输入矢量和这两个神经元权值矢量间的距离，window 为学习参数中的窗宽度参数。在上述条件下，如果神经元 i 输出为高时将产生正确的分类结果，神经元 j 输出为高时将产生错误的分类结果，那么这两个神经元的权值调整量为

dW (i , k) = lr * (P (k) - W (i, k))

dW (j , k) = -lr * (P (k)　W (i, k))

上式中lr为学习速率，P为神经元输入，W为当前权值矩阵。

函数调用时各输入量和返回量的含义参见函数 learngd。梯度矢量 gA 用来表示该层神经元的分类结果是否正确。该函数的学习参数包括：

- LP.lr：学习速率，缺省值为 0.01；
- LP.window：窗宽度，取值可以是 0 和 1 间的任意实数，缺省值为 0.25，通常取值在 0.2 和 0.3 之间。

例 3.34　对于一个两输入三神经元的网络层利用 learnlv2 函数进行学习，首先设定权值 W、输入矢量 P、梯度矢量 gA，并计算神经元的加权输入量 N 和输出量 A：



```
W = [ -0.75   1 ; 1   0 ; 1   1 ];
P = [ 0 ; 1 ];
gA = [ -1; 1; 1 ];
N = negdist ( W, P );
A = compet ( N );
```

然后设置学习速率并更新权值：

```
LP.lr = 0.5;   LP.window = 0.25;
dW = learnlv2 ( W, P, [ ], N, A, [ ], [ ], [ ], gA, [ ], LP, [ ] )

dW =
          -0.3750        0
           0             0
          -0.5000        0
```

网络属性设置：

如果要使网络第 i 层神经元的权值利用 learnlv2 函数进行学习，可作如下设置：

(1) 将网络的训练函数 net.trainFcn 设置为'trainr'，并设置相应的训练参数；

(2) 将网络的自适应调整函数 net.adaptFcn 设置为'trains'，并设置相应的自适应调整参数；

(3) 将网络第 i 层神经元的输入权值学习函数 net.inputWeights{i,j}.learnFcn 和网络权值学习函数 net.layerWeights{i,j}.learnFcn 设置为'learnlv2'，并设置相应的学习参数。

参见： learnlv1、adapt 和 train 函数。

10. learnos

功能： 星外权值学习函数。

格式：

① [dW, LS] = learnos (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

② info = learnos (code)

说明：

learnos 函数利用星外联想学习方法对权值进行调整，该方法中权值调整量为

dW (i, j) = lr * (A (i) - W (i, j)) * P (j)

其中，lr为学习速率，P为神经元输入，A为输出，W为当前权值矩阵。

函数调用时各输入量和返回量的含义参见函数 learngd。该函数的学习参数为学习速率 LP.lr，缺省值为 0.01。

例 3.35 根据下列输入矢量 P、输出矢量 A 和权值矩阵 W，利用 learnos 函数对神经元权值进行学习：

```
P = [ 0.2722; 0.1988; 0.0153 ];
A= [ 0.7468; 0.4451 ];
W = [ 0.6721    0.0196    0.3795;  0.8381    0.6813    0.8318];
```



设置学习速率并更新权值：

```
LP.lr = 0.5;
dW = learnos (W, P, [ ], [ ], A, [ ], [ ], [ ], [ ], [ ], LP, [ ])
dW =
        0.0102     0.0723     0.0028
       -0.0535    -0.0235    -0.0030
```

网络属性设置：

如果要使网络第 *i* 层神经元的权值利用 learnos 函数进行学习，可作如下设置：

(1) 将网络的训练函数 net.trainFcn 设置为'trainr'，并设置相应的训练参数；

(2) 将网络的自适应调整函数 net.adaptFcn 设置为'trains'，并设置相应的自适应调整参数；

(3) 将网络第 *i* 层神经元的输入权值学习函数 net.inputWeights{i,j}.learnFcn 和网络权值学习函数 net.layerWeights{i,j}.learnFcn 设置为'learnos'，并设置相应的学习参数。

参见： learnk、learnis、adapt 和 train 函数。

11. learnp

功能： 感知器权值和阈值学习函数。

格式：

① [dW, LS] = learnp (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

② [dB, LS] = learngd (B, P, Z, N, A, T, E, gW, gA, D, LP, LS)

③ info = learnp (code)

说明：

learnp 函数可用来对感知器网络的权值和阈值进行调整，该方法中权值调整量由输入矢量 P 和误差矢量 E 决定。

dW (i, j) = E (i) * P (j)

函数调用时各参数的含义参见函数 learncon 和 learngd。该函数没有学习参数。

例 3.36 一个三输入两神经元网络层的输入矢量 P 和误差矢量 E 分别为

```
P = [ 0.1509;  0.6979;  0.3784];
E = [ 1; -1 ];
```

使用learnp函数对权值进行更新：

```
dW = learnp ( [ ], P, [ ], [ ], [ ], [ ], E, [ ], [ ], [ ], [ ], [ ] )
dW =
        0.1509     0.6979     0.3784
       -0.1509    -0.6979    -0.3784
```

网络属性设置：

newp 函数建立的网络可以采用 learnp 函数作为学习函数，如果要使自定义网络第 *i* 层神经元的权值和阈值利用 learnp 函数进行学习，可作如下设置：

(1) 将网络的训练函数 net.trainFcn 设置为'trainb'，并设置相应的训练参数；





(2) 将网络的自适应调整函数 net.adaptFcn 设置为'trains'，并设置相应的自适应调整参数；

(3) 将网络第 i 层神经元的输入权值学习函数 net.inputWeights{i,j}.learnFcn、网络权值学习函数 net.layerWeights{i,j}.learnFcn 和阈值学习函数 net.biases{i}.learnFcn 设置为'learnp'，并设置相应的参数。

参见：learnpn、newp, adapt 和 train 函数。

12. learnpn

功能：归一化感知器权值和阈值学习函数。

格式：

① [dW, LS] = learnpn (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

② [dB, LS] = learngd (B, P, Z, N, A, T, E, gW, gA, D, LP, LS)

③ info = learnpn (code)

说明：

learnpn 函数多用于感知器网络的权值和阈值调整，当输入矢量中存在奇异样本时，该方法可以有效地提高感知器网络的训练速度。在该函数中，权值调整量由归一化的输入矢量 P 和误差矢量 E 共同决定：

dW (i, j) = E (i) * P (j) / nP

其中，nP为输入矢量P的模值：

nP = sqrt (1 + P(1)^2 + P(2)^2 + … + P(R)^2)

函数各输入量和返回量的含义参见函数 learncon 和 learngd。该函数没有学习参数。

例 3.37 一个三输入两神经元网络层的输入矢量 P 和误差矢量 E 分别为

```
P = [ 0.1509;  0.6979;  0.3784];
E = [ 1; -1 ];
```

利用learnpn函数对权值进行更新：

```
dW = learnpn ( [ ], P, [ ], [ ], [ ], [ ], E, [ ], [ ], [ ], [ ], [ ] )
dW =
        0.1173     0.5428     0.2943
       -0.1173    -0.5428    -0.2943
```

网络属性设置：

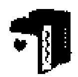
newp 函数建立的网络可以采用 learnpn 函数作为学习函数，如果要使自定义网络第 i 层神经元的权值和阈值利用 learnpn 函数进行学习，可作如下设置：

(1) 将网络的训练函数 net.trainFcn 设置为'trainb'，并设置相应的训练参数；

(2) 将网络的自适应调整函数 net.adaptFcn 设置为'trains'，并设置相应的适应调整参数；

(3) 将网络第 i 层神经元的输入权值学习函数 net.inputWeights{i,j}.learnFcn、网络权值学习函数 net.layerWeights{i,j}.learnFcn 和阈值学习函数 net.biases{i}.learnFcn 设置为'learnpn'，并设置相应的学习参数。

参见：learnp、newp, adapt 和 train 函数。



13. learnsom

功能： 自组织映射网络权值学习函数。

格式：

① [dW, LS] = learnsom (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

② info = learnsom (code)

说明：

learnsom 函数用于自组织映射网络中神经元权值的学习。在 learnsom 函数中，不仅获胜神经元的权值要作调整，获胜神经元邻域内的权值也要更新，即

dW (i, k) = lr * a2 * (P (k) - W (i, k))

其中，lr为学习速率，P为神经元输入，W为当前权值，a2为活动系数。第 i 个神经元的活动系数a2(i)按照如下方式确定：

a2 (i) = 1　如果第 i 个神经元获胜

a2 (i) = 0.5　如果第 j 个神经元获胜，并且和第 i 个神经元的间距小于邻域半径 nd。

a2 (i) = 0　其他。

函数调用时各输入量和返回量的含义参见函数 learngd 。该函数的学习参数包括：

- LP.order_lr：排列阶段学习速率，缺省值为 0.9 ；
- LP.order_steps：排列阶段的学习次数，缺省值为 1000 ；
- LP.tune_lr：调整阶段学习速率，缺省值为 0.02 ；
- LP.tune_nd：调整阶段邻域半径，缺省值为 1。

在本函数中，学习分排列和调整两阶段进行，学习速率 lr 在两个阶段中要从 LP.order_lr 降低到 LP.tune_lr，邻域半径 nd 要从神经元间的最大距离降低到 LP.tune_nd。

例 3.38　构造一个单层两输入九神经元的自组织映射网络，网络的拓扑结构为六边形，神经元间距采用 linkdist 计算方法。

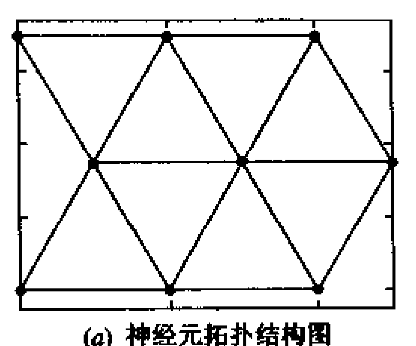
随机产生输入矢量 P、权值矩阵 W 和输出矢量 A：

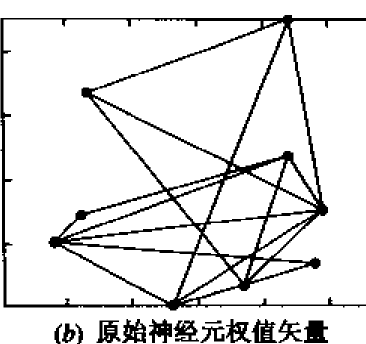
```
Pos = hextop ( 3, 3 );
plotsom ( Pos );          % 绘制神经元拓扑结构图，如图 3.15(a)所示
D = linkdist ( Pos );
P = rand (2, 1) ;
W = rand (9, 2) ;
plotsom ( W, D );         % 绘制原始神经元权值矩阵，如图 3.15(b)所示
A = rand (9, 1) ;
```

初始学习状态设为 [ ]，学习速率取为缺省值，并更新权值：

```
LP.order_lr = 0.9; LP.order_steps = 1000;
LP.tune_lr = 0.02; LP.tune_nd = 1;
LS=[ ];
[dW, LS] = learnsom (W, P, [ ], [ ], A, [ ], [ ], [ ], [ ], D, LP, LS);
plotsom ( W+dW, D );      % 绘制更新后的神经元权值矩阵，如图 3.15(c)所示。
```




(a) 神经元拓扑结构图


(b) 原始神经元权值矢量

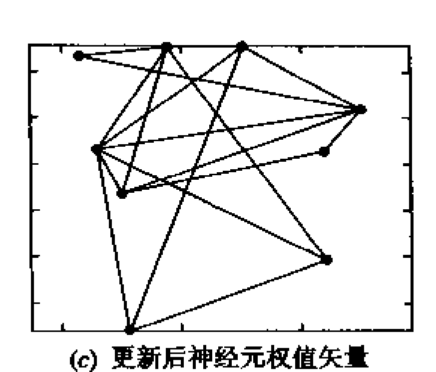

(c) 更新后神经元权值矢量

图 3.15 自组织映射网络神经元拓扑结构图和权值矩阵图

网络属性设置：

newsom 函数建立的网络采用 learnsom 函数作为学习函数，如果要使自定义网络第 *i* 层神经元的权值利用 learnsom 函数进行学习，可作如下设置：

(1) 将网络的训练函数 net.trainFcn 设置为'trainr'，并设置相应的训练参数；

(2) 将网络的自适应调整函数 net.adaptFcn 设置为'trains'，并设置相应的自适应调整参数；

(3) 将网络第 *i* 层神经元的输入权值学习函数 net.inputWeights{i,j}.learnFcn 和网络权值学习函数 net.layerWeights{i,j}.learnFcn 设置为'learnsom'，并设置相应的学习参数。

参见：adapt 和 train 函数。

14. learnwh

功能： Widrow-Hoff 权值和阈值学习函数。

格式：

① [dW, LS] = learnwh (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)

② [dB, LS] = learngd (B, P, Z, N, A, T, E, gW, gA, D, LP, LS)

③ info = learnwh (code)

说明：

learnwh 利用 Widrow-Hoff 规则对网络的权值和阈值进行学习，该函数多用于线性网络。在该方法中权值调整量由输入矢量 P、误差矢量 E 和学习速率 lr 共同确定：

dW (i，j) = lr * E (i) * P (j)

函数调用时各输入量和返回量的含义参见函数 learncon 和 learngd。该函数学习参数是由学习速率 LP.lr 构成的，缺省值为 0.01。

例 3.39 构造一个三输入两神经元网络层，输入矢量 P 和误差矢量 E 分别为

```
P = [ 0.5298;   0.6405;    0.2091 ];
E = [ 0.3798;   0.7833];
```

设定学习速率，并利用 learnwh 函数对权值进行学习：

```
LP.lr = 0.5 ;
dW = learnwh ([ ], P, [ ], [ ], [ ], [ ], E, [ ], [ ], [ ], LP, [ ])
```



dW =

0.1006	0.1216	0.0397
0.2075	0.2509	0.0819

网络属性设置：

newlin 函数建立的网络可以采用 learnwh 函数作为学习函数，如果要使自定义网络第 *i* 层神经元的权值和阈值利用 learnwh 函数进行学习，可作如下设置：

(1) 将网络的训练函数 net.trainFcn 设置为'trainb'，并设置相应的训练参数；

(2) 将网络的自适应调整函数 net.adaptFcn 设置为'trains'，并设置相应的自适应调整参数；

(3) 将网络第 *i* 层神经元的输入权值学习函数 net.inputWeights{i,j}.learnFcn、网络权值学习函数 net.layerWeights{i,j}.learnFcn 和阈值学习函数 net.biases{i}.learnFcn 设置为'learnwh'，并设置相应的学习参数。

参见： newlin、adapt 和 train 函数。

3.2.8 性能函数

网络的性能函数如表 3-9 所示。性能函数通过计算网络输出矢量和目标矢量之间的差异来衡量神经网络的性能。

网络属性设置：

神经网络性能函数的设置步骤如下：

(1) 将网络的性能函数 net.performFcn 设置为期望的性能函数；

(2) 设置相应的性能函数参数 net.performParam。

表 3-9　网络性能函数

函数名称	功　　能
mae	平均绝对误差函数
mse	均方误差函数
msereg	规则化均方误差函数
sse	平方和误差函数

1. mae

功能： 平均绝对误差函数。

格式：

① perf = mae (E, X, PP)

② perf = mae (E, net, PP)

③ info = mae (code)

说明：

mae 函数用来计算输出量和目标矢量之间的平均绝对误差。

在函数的调用形式①中，输入 E 为误差矢量；X 是由网络权值和阈值参数构成的矩



阵；PP 为性能参数。mae 函数在调用时，输入 X 和 PP 可省略。在函数的调用形式②中，输入 net 为神经网络对象，函数可以从对象中获得网络的权值和阈值。在调用形式②中，输入 net 也可被省略。

函数的调用形式③返回该性能函数的有关信息，code 字符串取值为：

- 'deriv'：返回性能函数导函数的名称；
- 'name'：返回性能函数的全称；
- 'pnames'：返回性能函数参数的名称；
- 'pdefaults'：返回性能函数参数的缺省值。

利用 newp 函数建立的感知器网络采用 mae 函数作为其性能函数。

例 3.40 计算下面误差矢量的平均绝对误差：

```
E = [ -0.4326;   -1.6656;    0.1253 ];
mae ( E )
ans =
          0.7412
```

参见：mse、msereg 和 dmae 函数。

2. mse

功能：均方误差函数。

格式：

① perf = mse (E, X, PP)

② perf = mse (E, net, PP)

③ info = mse (code)

说明：

mse 函数用来计算输出矢量和目标矢量之间的均方误差。函数调用时各输入量和返回量的含义参见 mae 函数。利用 newcf、newff 和 newelm 函数建立的网络可以采用 mse 函数作为其性能函数。

例 3.41 计算下面误差矢量的均方误差：

```
E = [ 0.2877;   -1.1465;   1.1909 ];
mse ( E )
ans =
         0.9385
```

参见：mae、msereg 和 dmse 函数。

3. msereg

功能：规则化均方误差函数。

格式：

① perf = msereg (E, X, PP)

② perf = msereg (E, net, PP)

③ info = msereg (code)

说明：

msereg 函数用来计算网络的规则化均方误差，该性能函数不仅考虑了网络输出的均方误差，还考虑了网络中各权值和阈值参数的均方和，网络的性能由这二者的加权和决定。函数调用时各输入量和返回量的含义参见 mae 函数。该函数的性能参数为：

- PP.ratio：误差函数中网络输出均方误差的权重因子，1-PP.ratio 则是权值和阈值参数均方和的权重因子。

由于 msereg 函数在调用时要使用网络参数和性能参数，因此函数输入 X、net 以及 PP 不能省略。

利用 newcf、newff 和 newelm 函数建立的网络可以采用 msereg 函数作为其性能函数。

例 3.42 根据下面的误差矢量 E、权值及阈值矩阵 X，计算规则化均方误差。

```
E = [ 1.1892;   -0.0376;   0.3273 ];
X= [ 0.4860      0.4565      0.4447;
     0.8913      0.0185      0.6154;
     0.7621      0.8214      0.7919 ];
```

设置权重因子，并计算规则化均方误差：

```
PP.ratio = 0.9;
msereg ( E, X, PP )
ans =
          0.5801
```

参见： mae、mse 和 dmsereg 函数。

4．sse

功能： 平方和误差函数。

格式：

① perf = sse (E, X, PP)

② perf = sse (E, net, PP)

③ info = sse (code)

说明：

sse 函数用来计算网络输出矢量和目标矢量之间的误差平方和。函数调用时各输入量和返回量的含义参见 mae 函数。

例 3.43 对下面的误差矢量计算平方和误差。

```
E = [0.1746;   -0.1867;   0.7258];
sse ( E )
ans =
          0.5921
```

参见： dsse 函数。

3.2.9 性能函数的导函数

网络性能函数的导函数如表 3-10 所示，这些函数用来求取性能函数的输出对于各权值



及阈值参数的导数，在程序中不直接调用。

表 3-10　性能函数的导函数

函数名称	功　　能
Dmae	平均绝对误差函数的导函数
Dmse	均方误差函数的导函数
Dmsereg	规则化均方误差函数的导函数
Dsse	平方和误差函数的导函数

1. dmae

功能：平均绝对误差函数的导函数。

格式：

① dPerf_dE = dmae ('e', E, X, perf, PP)

② dPerf_dX = dmae ('x', E, X, perf, PP)

说明：

dmae 函数用来计算平均绝对误差函数的输出对其输入的导数。

在函数的调用形式①、②中，输入 E 为误差矢量；X 是由网络权值和阈值参数构成的矩阵；perf 为网络的性能值；PP 为性能参数。调用形式①返回网络性能对于网络输出误差的导数 dPerf_dE，调用形式②返回网络性能对于各权值和阈值的导数 dPerf_dX。

例 3.44　设网络的输出误差矢量 E 和权值矢量 X 为

```
E = [ 1; -0.5 ];
X = [ 3.5; 0.2; -2.2; 4.1 ];
```

首先计算平均绝对误差：

```
perf = mae ( E )
perf =
          0.7500
```

然后计算平均绝对误差对网络输出误差和权值的导数：

```
dPerf_dE = dmae ( 'e', E, X )
dPerf_dE =
          [2x1 double]
dPerf_dX = dmae ( 'x', E, X );
dPerf_dX' =
ans =
          0     0     0     0
```

dPerf_dE返回了一个单元数组，即

```
dPerf_dE{1}'
ans =
          1     -1
```

参见：mae 函数。



2. dmse

功能：均方误差函数的导函数。

格式：

① dPerf_dE = dmse ('e', E, X, perf, PP)

② dPerf_dX = dmse ('x', E, X, perf, PP)

说明：

dmse 函数用来计算均方误差函数的输出对其输入的导数。函数调用时各输入量和返回量的含义参见 dmae 函数。

例 3.45 根据下列误差矢量 E 和权值矢量 X 计算网络的均方误差及其导数。

```
E = [ 1; -0.5 ];
X = [ 3.5; 0.2; -2.2; 4.1 ];
perf = mse ( E )
perf =
            0.6250
dPerf_dE = dmse ( 'e', E, X )
dPerf_dE =
            [2x1 double]
dPerf_dE{1}'
ans =
           1.0000    -0.5000
dPerf_dX = dmse ( 'x', E, X );
dPerf_dX' =
ans =
            0        0        0        0
```

参见：mse 函数。

3. dmsereg

功能：规则化均方误差函数的导函数。

格式：

① dPerf_dE = dmsereg ('e', E, X, perf, PP)

② dPerf_dX = dmsereg ('x', E, X, perf, PP)

说明：

dmsereg 函数用来计算规则化均方误差函数的输出对其输入的导数。性能参数 PP 参见函数 msereg，函数调用时其余各输入量和返回量的含义参见 dmae 函数。

例 3.46 对下列误差矢量 E 和权值矢量 X 计算规则化均方误差及其导数。

```
E = [ 1; -0.5 ];
X = [ 3.5; 0.2; -2.2; 4.1 ];
PP.ratio = 0.9;
perf = msereg ( E, X, PP )
```



```
perf =
            1.4110
dPerf_dE = dmsereg ( 'e', E, X, perf, PP )
dPerf_dE =
            [2x1 double]
dPerf_dE{1}'
ans =
          0.9000    -0.4500
dPerf_dX = dmsereg ( 'x', E, X, perf, PP );
dPerf_dX' =
ans =
          -0.1750    -0.0100    0.1100    -0.2050
```

参见：msereg 函数。

4. dsse

功能：平方和误差函数的导函数。

格式：

① dPerf_dE = dsse ('e', E, X, perf, PP)

② dPerf_dX = dsse ('x', E, X, perf, PP)

说明：

dsse 函数用来计算平方和误差函数的输出对其输入的导数。函数调用时各参数的含义参见 dmae 函数。

例 3.47 对下列误差矢量 E 和权值矢量 X 计算平方和误差及其导数。

```
E = [ 1; -0.5 ];
X = [ 3.5; 0.2; -2.2; 4.1 ];
perf = sse ( E )
perf =
            1.2500
dPerf_dE = dsse ( 'e', E, X )
dPerf_dE =
            [2x1 double]
dPerf_dE{1}'
ans =
          2       -1
dPerf_dX = dsse ( 'x', E, X );
dPerf_dX' =
ans =
          0       0       0       0
```

参见：dsse 函数。

3.2.10 线搜索函数

网络训练时采用的线搜索函数如表3-11所示。线搜索函数主要用于一些准牛顿反向传播算法和共轭梯度反向传播算法中，采用这些算法进行训练的网络，如函数 newcf、newff 以及 newelm 建立的网络在训练时都要用到线搜索函数。

网络属性设置：

如果要设置网络训练算法中的线搜索函数，可按如下步骤进行：

(1) 将网络的训练函数 net.trainFcn 设置为采用准牛顿算法和共轭梯度算法的训练函数，如 traincgf、traincgb、traincgp、trainbfg 和 trainoss 等；

(2) 将训练参数中的 net.trainParam.searchFcn 属性设置为期望的线搜索函数；

(3) 设置训练参数 net.trainParam 中其他参数的属性。

表3-11 线搜索函数

函数名称	功 能
srchbac	回溯线搜索方法
srchbre	Brent 线搜索方法
srchcha	Charalambous 线搜索方法
srchgol	Golden 区间分割线搜索方法
Srchhyb	二分和立方插值混合线搜索方法

1. srchbac

功能： 回溯线搜索方法。

格式：

```
[ a, gX, perf, retcode, delta, tol ] =
srchbac ( net, X, Pd, Tl, Ai, Q, TS, dX, gX, perf, dperf, delta, tol, ch_perf )
```

说明：

srchbac 函数利用回溯法寻找性能曲面的最低点，该函数适用于各种准牛顿算法。

函数的输入 net 为神经网络对象；X 是由当前权值和阈值参数构成的矩阵；Pd 为延迟输入矢量；Tl 为每层的目标矢量；Ai 为初始输入延迟条件；Q 为批处理数据的个数；TS 为网络工作的时间长度；dX 为表示搜索方向的矢量；gX 为梯度矢量；perf 为性能曲面在当前权值和阈值矢量 X 处的取值；dperf 为性能曲面在 X 点沿 dX 方向的斜率；delta 为初始搜索步长；tol 为搜索误差；ch_perf 是在先前步长下性能函数的变化值。在函数返回值中，a 为性能函数从当前点到达最低点要经过的步长；gX 为性能曲面在最低点处的梯度；perf 为性能函数在最低点处的取值；delta 为新的初始搜索步长；tol 为新的搜索误差；retcode 返回函数执行过程中的一些相关信息。retcode 由三个元素组成，其中前两个元素分别返回搜索过程两个阶段中进行网络性能评估的次数，第三个元素的返回值可能取下列各值：

- 0：表示函数运行正常；

- 1：表示已达到了最小步长；
- 2：表示已达到了最大步长；
- 3：表示未满足 beta 条件。

回溯搜索方法用到的参数包括：

- scal_tol：确定线搜索误差的尺度因子，初始步长 delta 除以该参数后得到线搜索结果的误差，常设为 20；
- alpha：确定性能函数下降是否足够大的尺度因子；
- beta：确定步长是否足够大的尺度因子；
- low_lim：步长变化下界；
- up_lim：步长变化上界；
- maxstep：最大步长；
- minstep：最小步长。

以上这些参数包含在训练函数的参数中，需要在设置训练参数的同时进行设置。

参见：srchgol、srchbre、srchcha 和 srchhyb 函数。

2. srchbre

功能：Brent 线搜索方法。

格式：

[a, gX, perf, retcode, delta, tol] =
srchbre (net, X, Pd, Tl, Ai, Q, TS, dX, gX, perf, dperf, delta, tol, ch_perf)

说明：

srchbre 函数利用 Brent 搜索方法寻找性能函数曲面的最低点，该方法结合了 Golden 区间分割搜索方法（参见 srchgol 函数）和二次插值算法的特点，其收敛性优于 Golden 搜索法。

函数调用时各输入量和返回量的含义参见 srchbac 函数，搜索时用到的参数如下：

- scal_tol：确定线搜索误差的尺度因子，初始步长 delta 除以该参数后得到线搜索结果的误差，常设为 20；
- alpha：确定性能函数下降是否足够大的尺度因子；
- beta：确定步长是否足够大的尺度因子；
- bmax：最大步长。

参见：srchgol、srchbac、srchcha 和 srchhyb 函数。

3. srchcha

功能：Charalambous 线搜索方法。

格式：

[a, gX, perf, retcode, delta, tol] =
srchcha (net, X, Pd, Tl, Ai, Q, TS, dX, gX, perf, dperf, delta, tol, ch_perf)

说明：

srchcha 函数利用 Charalambous 线搜索方法寻找性能函数曲面的最低点，该方法是结合区间分割和三次插值算法的混合算法。



函数调用时各输入量和返回量的含义参见 srchbac 函数，搜索时用到的参数如下：

- scal_tol：确定线搜索误差的尺度因子，初始步长 delta 除以该参数后得到线搜索结果的误差，常设为 20 ；
- alpha：确定性能函数下降是否足够大的尺度因子；
- beta：确定步长是否足够大的尺度因子；
- gama：避免性能函数下降过小的参数，常设为 0.1。

参见：srchgol、srchbac、srchbre 和 srchhyb 函数。

4. srchgol

功能：Golden 区间分割线搜索方法。

格式：

[a, gX, perf, retcode, delta, tol] =

srchgol (net, X, Pd, Tl, Ai, Q, TS, dX, gX, perf, dperf, delta, tol, ch_perf)

说明：

srchgol 函数利用 Golden 区间分割方法寻找性能曲面的最低点，这种方法通过对搜索区间的不断分割和定位，最终将最低点定位在一个足够小的区间内，这个区间的长度也就是搜索的误差。

函数调用时各输入量和返回量的含义参见 srchbac 函数，搜索时用到的参数如下：

- scal_tol：确定线搜索误差的尺度因子，初始步长 delta 除以该参数后得到线搜索结果的误差，常设为 20 ；
- alpha：确定性能函数下降是否足够大的尺度因子；
- bmax：最大步长。

参见：srchbre、srchbac、srchcha 和 srchhyb 函数。

5. srchhyb

功能：二分和立方插值混合线搜索方法。

格式：

[a, gX, perf, retcode, delta, tol] =

srchhyb (net, X, Pd, Tl, Ai, Q, TS, dX, gX, perf, dperf, delta, tol, ch_perf)

说明：

srchhyb 函数利用一种混合算法寻找性能曲面的最低点，该算法结合了二分法和立方插值算法的特点。

函数调用时各输入量和返回量的含义参见 srchbac 函数，搜索时用到的参数如下：

- scal_tol：确定线搜索误差的尺度因子，初始步长 delta 除以该参数后得到线搜索结果的误差，常设为 20 ；
- alpha：确定性能函数下降是否足够大的尺度因子；
- beta：确定步长是否足够大的尺度因子；
- bmax：最大步长。

参见：srchgol、srchbac、srchcha 和 srchbre 函数。



3.2.11 传递函数

神经元的传递函数如表 3-12 所示。

网络属性设置：

如果要设置网络第 i 层神经元的传递函数，只要把该层的传递函数属性 net.layers{i}.transferFcn 设为期望的传递函数即可。

表 3-12 传递函数

函数名称	功能
compet	竞争传递函数
Hardlim	硬限幅传递函数
Hardlims	对称硬限幅传递函数
Logsig	对数 Sigmoid 传递函数
Poslin	正线性传递函数
Purelin	纯线性传递函数
Radbas	高斯径向基传递函数
Satlin	饱和线性传递函数
Satlins	对称饱和线性传递函数
Softmax	softmax 传递函数
Tansig	正切 Sigmoid 传递函数
Tribas	三角基传递函数

1. compet

功能： 竞争传递函数。

格式：

① A = compet (N)

② info = compet (code)

说明：

竞争传递函数 compet 用来寻找输入矢量 N 中的最大元素，并把相应神经元的输出设为 1，其余输出设为 0。输出为 1 的神经元称为获胜神经元。函数示意图如图 3.16 所示。

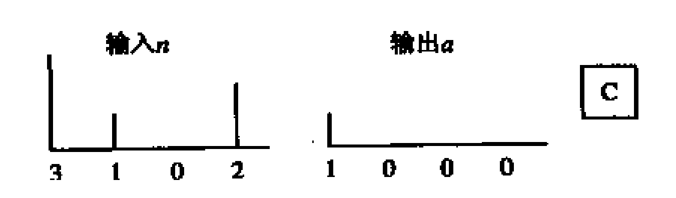


图 3.16 竞争函数示意图



函数调用形式②返回有关函数的信息，code 字符串取值为：

- 'deriv'：返回导函数的名称，函数 compet 没有导函数；
- 'name'：返回传递函数的全称；
- 'output'：返回传递函数的输出范围；
- 'active'：返回传递函数的活跃输入范围。

newc 和 newpnn 函数建立的网络都采用 compet 函数作为传递函数。

例 3.48　求竞争传递函数对下列输入矢量的输出：

```
n = [ 0, 1; -0.5, 0.5 ];
a = compet ( n )
a =
        (1,1)        1
        (1,2)        1
```

参见：sim 和 softmax 函数。

2. hardlim

功能：硬限幅传递函数。

格式：

① A = hardlim (N)

② info = hardlim (code)

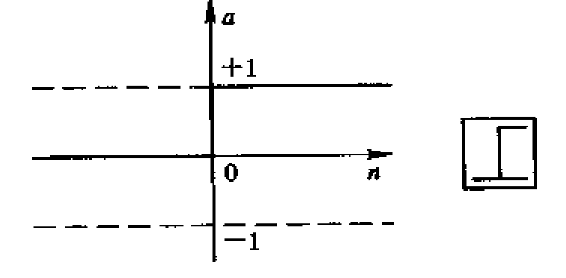


图 3.17　硬限幅传递函数示意图

说明：

硬限幅传递函数 hardlim 把函数输入矢量 N 中各元素的取值强迫限制为 1 或 0，当输入大于或等于 0 时，神经元的输出为 1，否则输出为 0。函数曲线如图 3.17 所示。

newp 函数建立的网络可以采用 hardlim 函数作为传递函数。

例 3.49　求硬限幅传递函数对下列输入矢量的输出。

```
n = [ 0; 1; -0.5 ];
a = hardlim ( n );
a'
ans =
```



```
        1    1    0
```

参见：sim 和 hardlims 函数。

3. hardlims

功能：对称硬限幅传递函数。

格式：

① A = hardlims (N)

② info = hardlims (code)

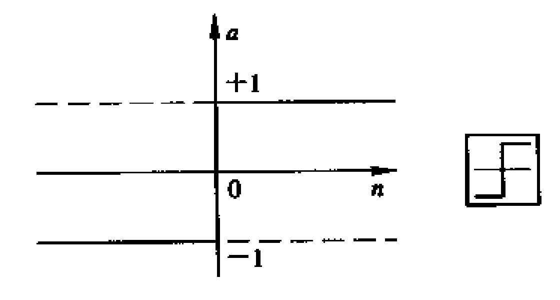


图 3.18　对称硬限幅传递函数示意图

说明：

对称硬限幅传递函数 hardlims 把函数输入矢量 N 中各元素的取值强迫限制为 1 或−1，当输入大于或等于 0 时，神经元的输出为 1，否则输出为−1。函数曲线如图 3.18 所示。

newp 函数建立的网络可以采用 hardlims 函数作为传递函数。



例 3.50　求对称硬限幅传递函数对下列输入矢量的输出。

```
n = [ 0; 1; -0.5 ];
a = hardlims ( n );
a'
ans =
        1      1      -1
```

参见：sim 和 hardlim 函数。

4. logsig

功能：对数 Sigmoid 传递函数。

格式：

① A = logsig (N)

② info = logsig (code)

说明：

对数 Sigmoid 传递函数的计算公式用 MATLAB 语句表示为

logsig (n) = 1 / (1 + exp(-n))

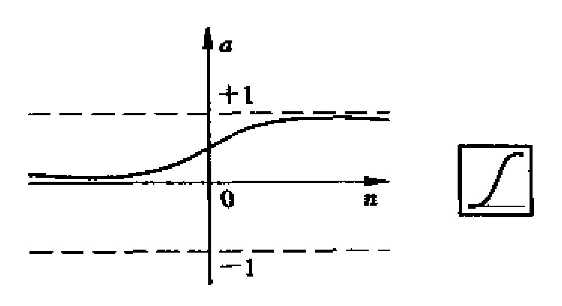


图 3.19　logsig 传递函数示意图

函数曲线如图3.19所示。

newff 和 newcf 函数建立的网络都可以采用该函数作为传递函数。

例 3.51　求对数 Sigmoid 传递函数对下列输入矢量的输出。

```
n = [ -1; 0; 0.5 ];
a = logsig ( n );
a'
ans =
        0.2689      0.5000      0.6225
```

参见：sim、dlogsig 和 tansig 函数。

5. poslin

功能：正线性传递函数。

格式：

① A = poslin (N)

② info = poslin (code)

说明：

对于输入矢量 N 中的正元素或 0，正线性传递函数按照原始数值输出；对于负元素，函数将输出 0。函数曲线如图 3.20 所示。

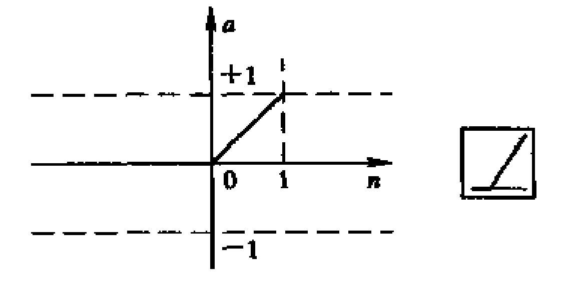


图 3.20　正线性传递函数示意图

例 3.52　求正线性传递函数对下列输入矢量的输出：

```
n = [ -1; 0; 0.5 ];
a = poslin ( n );
a'
```



```
ans =
          0     0     0.5
```

参见：sim、purelin、satlin 和 satlins 函数。

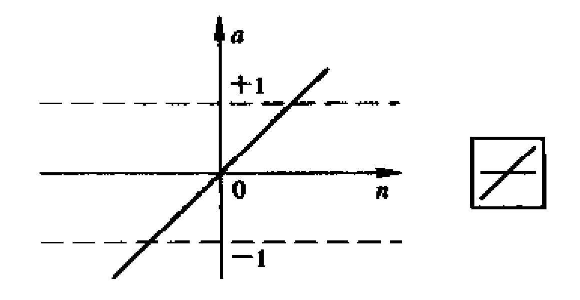


图3.21　纯线性传递函数示意图

6. purelin

功能：纯线性传递函数。

格式：

① A = purelin (N)

② info = purelin (code)

说明：

纯线性传递函数把输入 N 按照原始数值输出。函数曲线如图 3.21 所示。newlin 和 newlind 函数建立的网络都可以采用该函数作为传递函数。

例 3.53　求纯线性传递函数对下列输入矢量的输出。

```
n = [ -1; 0; 0.5 ];
a = purelin ( n );
a'
ans =
          -1     0     0.5
```

参见：sim、dpurelin、satlin 和 satlins 函数。

7. radbas

功能：高斯径向基传递函数。

格式：

① A = radbas (N)

② info = radbas (code)

说明：

高斯径向基传递函数的计算公式用 MATLAB 语句表示为

```
a = exp ( -n.^2 )
```

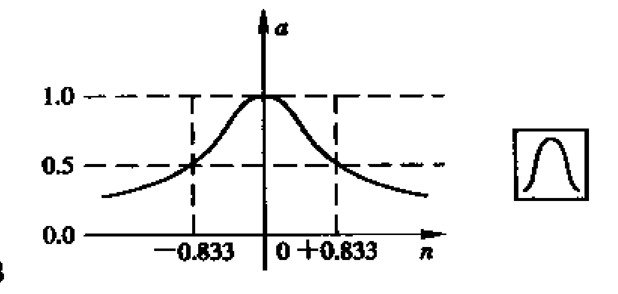


图 3.22　高斯径向基传递函数示意图

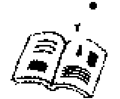
函数曲线如图3.22所示。

newpnn和newgrnn函数建立的网络都可以采用该函数作为传递函数。

例 3.54　求高斯径向基传递函数对下列输入矢量的输出：

```
n = [ -1; 0; 0.5 ];
a = radbas ( n );
a'
ans =
          0.3679     1.0000     0.7788
```

参见：sim、dradbas 和 tribas 函数。



8. satlin

功能：饱和线性传递函数。

格式：

① A = satlin (N)

② info = satlin (code)

说明：

饱和线性函数把输入矢量 N 中位于[0, 1]区间内的元素按照原始数值输出；对于负元素，则输出 0；对于大于 1 的元素，则输出 1。函数曲线如图 3.23 所示。

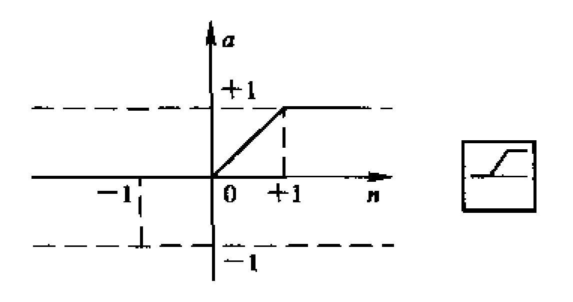


图 3.23　饱和线性传递函数示意图

例 3.55　求饱和线性传递函数对下列输入矢量的输出：

```
n = [ -1; 0; 0.5 ; 2 ];
a = satlin ( n );
a'
ans =
          0          0     0.5000     1.0000
```

参见：sim、poslin、purelin 和 satlins 函数。

9. satlins

功能：对称饱和线性传递函数。

格式：

① A = satlins (N)

② info = satlins (code)

说明：

对称饱和线性传递函数把输入矢量 N 中位于[-1, 1]区间内的元素按照原始数值输出；对于小于-1 的元素，则输出-1；对于大于 1 的元素，则输出 1。函数曲线如图 3.24 所示。

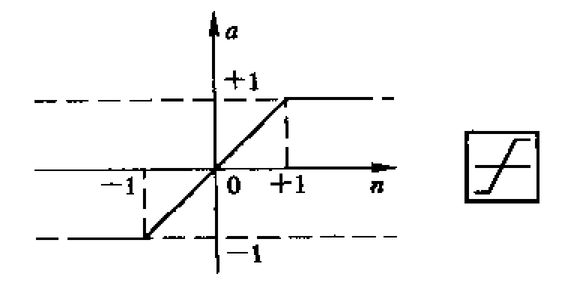


图 3.24　对称饱和线性传递函数示意图

newhop 函数建立的网络采用该函数作为传递函数。

例 3.56　求对称饱和线性传递函数对下列输入矢量的输出：

```
n = [ -3; -1; 0; 0.5 ; 2 ];
a = satlins ( n );
a'
ans =
    -1.0000    -1.0000          0     0.5000     1.0000
```

参见：sim、poslin、purelin 和 satlin 函数。

10. softmax

功能：softmax 传递函数。

格式:

① A = softmax (N)

② info = softmax (code)

说明:

softmax 传递函数的计算公式用 MATLAB 语句表示为

a = exp (n) / sum (exp (n))

函数曲线如图3.25所示。该函数没有导函数。

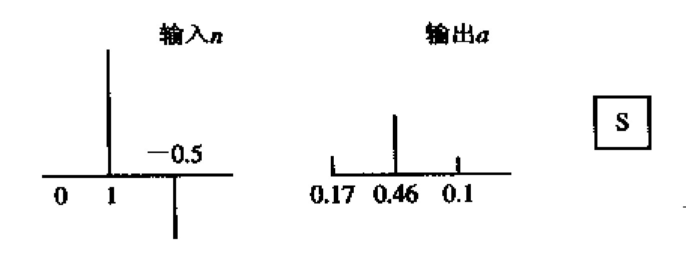


图 3.25 softmax 传递函数示意图

例 3.57 求 softmax 传递函数对下列输入矢量的输出:

```
n = [  -1;   0.5 ; 2 ];
a = softmax ( n );
a'
ans =
          0.0391      0.1753      0.7856
```

参见: sim 和 compet 函数。

11. tansig

功能: 正切 Sigmoid 传递函数。

格式:

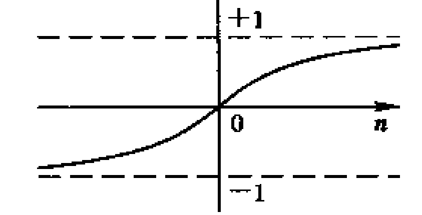
① A = tansig (N)

② info = tansig (code)

说明:

tansig 传递函数的计算公式用 MATLAB 语句表示为

n = 2/ (1+ exp (-2*n)) -1



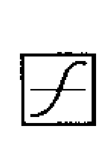


图 3.26 tansig 传递函数示意图

函数曲线如图3.26所示。newff和newcf函数建立的网络都可以采用该函数作为传递函数。

例 3.58 求 tansig 传递函数对下列输入矢量的输出。

```
n = [-1;   0.5 ; 2 ];
a = tansig ( n );
a'
ans =
          -0.7616      0.4621      0.9640
```



参见：sim、dtansig 和 logsig 函数。

12. tribas

功能：三角基传递函数。

格式：

① A = tribas (N)

② info = tribas (code)

说明：

当输入矢量 N 中的元素位于[-1, 1]区间内时，tribas 传递函数的计算公式用 MATLAB 语句表示为

n = 1−abs (n)

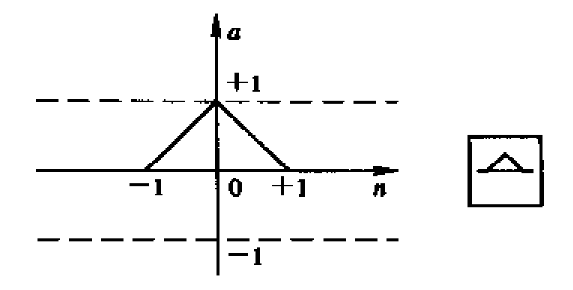


图 3.27 tribas 传递函数示意图

当元素在该区间之外时，函数输出为0。函数曲线如图3.27所示。

例 3.59 求 tribas 传递函数对下列输入矢量的输出。

```
n = [  -1;  -0.5 ; 0.9;  2 ];
a = tribas ( n );
a'
ans =
          0    0.5000    0.1000    0
```

参见：sim 和 radbas 函数。

3.2.12 传递函数的导函数

神经元传递函数的导函数如表 3-13 所示。

表 3-13 传递函数的导函数

函数名称	功能
Dhardlim	硬限幅传递函数的导函数
Dhardlms	对称硬限幅传递函数的导函数
Dlogsig	对数 Sigmoid 传递函数的导函数
dposlin	正线性传递函数的导函数
dpurelin	纯线性传递函数的导函数
dradbas	高斯径向基传递函数的导函数
dsatlin	饱和线性传递函数的导函数
dsatlins	对称饱和线性传递函数的导函数
dtansig	tansig 传递函数的导函数
dtribas	三角基传递函数的导函数

1. dhardlim

功能：硬限幅传递函数的导函数。

格式：

dA_dN = dhardlim (N, A)

说明：

该函数用来计算硬限幅函数的输出 A 相对于输入 N 的导数。

例 3.60　求硬限幅传递函数的输出对下列输入矢量的导数。

```
n = [ 0; 1; -0.5 ];
a = hardlim ( n );
da_dn = dhardlim ( n, a );
da_dn'
ans =
        0    0    0
```

参见：hardlim 函数。

2. dhardlims

功能：对称硬限幅传递函数的导函数。

格式：

dA_dN = dhardlims (N, A)

说明：

该函数用于计算对称硬限幅函数的输出 A 相对于输入 N 的导数。

例 3.61　求对称硬限幅传递函数的输出对下列输入矢量的导数。

```
n = [ 0; 1; -0.5 ];
a = hardlims ( n );
da_dn = dhardlms ( n, a );
da_dn'
ans =
        0    0    0
```

参见：hardlims 函数。

3. dlogsig

功能：对数 Sigmoid 传递函数的导函数。

格式：

dA_dN = dlogsig (N, A)

说明：

该函数用于计算对数 Sigmoid 函数的输出 A 相对于输入 N 的导数。

例 3.62　求对数 Sigmoid 函数的输出对下列输入矢量的导数。

```
n = [ -1; 0; 0.5 ];
a = logsig ( n );
```



```
da_dn = dlogsig ( n, a );
da_dn'
ans =
        0.1966    0.2500    0.2350
```

参见：logsig、tansig 和 dtansig 函数。

4. dposlin

功能： 正线性传递函数的导函数。

格式：

dA_dN = dposlin (N, A)

说明：

该函数用于计算正线性函数的输出 A 相对于输入 N 的导数。

例 3.63 求正线性传递函数的输出对下列输入矢量的导数。

```
n = [ -1; 0; 0.5 ];
a = poslin ( n );
da_dn = dposlin ( n, a );
da_dn'
ans =
        0    1    1
```

参见：poslin 函数。

5. dpurelin

功能： 纯线性传递函数的导函数。

格式：

dA_dN = dpurelin (N, A)

说明：

该函数用于计算纯线性函数的输出 A 相对于输入 N 的导数。

例 3.64 求纯线性传递函数的输出对下列输入矢量的导数。

```
n = [ -1; 0; 0.5 ];
a = purelin ( n );
da_dn = dpurelin ( n, a );
da_dn'
ans =
        1    1    1
```

参见：purelin 函数。

6. dradbas

功能： 高斯径向基传递函数的导函数。



格式：

dA_dN = dradbas (N, A)

说明：

该函数用于计算高斯径向基函数的输出 A 相对于输入 N 的导数。

例 3.65　求高斯径向基传递函数的输出对下列输入矢量的导数。

```
n = [ -1; 0; 0.5 ];
a = radbas ( n );
da_dn = dradbas ( n,a );
da_dn'
ans =
            0.8734      1.0000      0.8593
```

参见： radbas 函数。

7. dsatlin

功能： 饱和线性传递函数的导函数。

格式：

dA_dN = dsatlin (N, A)

说明：

该函数用于计算饱和线性函数的输出 A 相对于输入 N 的导数。

例 3.66　求饱和线性传递函数的输出对下列输入矢量的导数。

```
n = [ -1; 0; 0.5 ; 2];
a = satlin ( n );
da_dn = dsatlin ( n， a );
da_dn'
ans =
            0        1        1        0
```

参见： satlin 函数。

8. dsatlins

功能： 对称饱和线性传递函数的导函数。

格式：

dA_dN = dsatlins (N, A)

说明：

该函数用于计算对称饱和线性函数的输出 A 相对于输入 N 的导数。

例 3.67　求对称饱和线性传递函数的输出对下列输入矢量的导数。

```
n = [ -3; -1; 0; 0.5 ; 2];
a = satlins ( n );
da_dn = dsatlins ( n, a );
da_dn'
```



```
ans =
          0     1     1     1     0
```

参见：satlins 函数。

9. dtansig

功能： tansig 传递函数的导函数。

格式：

dA_dN = dtansig (N, A)

说明：

该函数用于计算 tansig 函数的输出 A 相对于输入 N 的导数。

例 3.68 求 tansig 传递函数的输出对下列输入矢量的导数。

```
n = [-1;  0.5 ; 2];
a = tansig ( n );
da_dn = dtansig ( n, a );
da_dn'
ans =
          0.4200     0.7864     0.0707
```

参见： tansig、logsig 和 dlogsig 函数。

10. dtribas

功能： 三角基传递函数的导函数。

格式：

dA_dN = dtribas (N, A)

说明：

该函数用于计算三角基函数的输出 A 相对于输入 N 的导数。

例 3.69 求 tribas 传递函数的输出对下列输入矢量的导数。

```
n = [ -1;  -0.5 ; 0.9;  2 ];
a = tribas ( n );
da_dn = dtribas ( n, a );
da_dn'
ans =
          1     1    -1     0
```

参见： tribas 函数

3.2.13 加权函数

加权函数如表 3-14 所示，这类函数确定了神经网络每层的输入矢量和神经元权值矩阵通过何种计算方式得到神经元传递函数的加权输入量。

表 3-14 加权函数

函数名称	功能
dist	欧氏距离加权函数
dotprod	内积加权函数
mandist	曼哈顿距离加权函数
negdist	负欧氏距离加权函数
normprod	规则化内积加权函数

1. dist

功能：欧氏距离加权函数。

格式：

① Z = dist (W, P)

② df = dist ('deriv')

③ D = dist (pos)

说明：

dist 函数用于计算两矢量间的欧氏距离，其计算原理用 MATLAB 语句表示为

D = sum ((x − y).^2).^0.5

上式中，D是矢量x和y之间的欧氏距离。

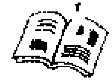
在函数的调用形式①中，W 为权值矩阵，P 为该层网络的输入矢量，此时函数返回加权输入矩阵 Z。函数的调用形式②返回距离函数导函数的名称，由于欧氏距离函数没有导函数，所以调用形式②将返回空字符串。函数的调用形式③用于计算自组织映射网络中神经元之间的距离，函数中 pos 为表示神经元空间位置的坐标矩阵，此时函数返回距离矩阵 D。

例 3.70 设定权值矩阵 W 和输入矢量 P，计算欧氏距离的加权输入。

```
W = [ 0.9   0.6   0.8; 0.2   0.5   0.7 ];
P = [ 0.4   0   0.8 ]';
Z = dist ( W, P )
Z =
     0.7810
     0.5477
```

例 3.71 设定三个神经元的位置坐标矢量 pos，计算它们之间的欧氏距离。

```
pos = [ 0.4   0.9   0.4; 0.6   0.7   0.9; 0.8   0.2   0.9 ];
D = dist ( pos )
D =
          0    0.7874    0.3162
     0.7874         0    0.8832
     0.3162    0.8832         0
```



网络属性设置：

newpnn 和 newgrnn 函数建立的网络都采用 dist 函数作为加权函数。如果要使网络第 *i* 层神经元权值的加权函数为 dist 函数，只要将该层输入权值和网络权值的加权函数 net.inputWeights{i,j}.weightFcn 及 net.layerWeights{i,j}.weightFcn 设置为'dist'即可。

如果要使网络第 *i* 层的距离函数为 dist 函数，只要将该层网络的距离函数 net.layers{i}.distanceFcn 设置为'dist'即可。

参见： sim、dotprod、negdist、normprod、mandist 和 linkdist 函数。

2. dotprod

功能： 内积加权函数。

格式：

① Z = dotprod (W, P)

② df = dotprod ('deriv')

说明：

dotprod 函数用于计算两矢量之间的内积。

在函数的调用形式①中，W 为权值矩阵，P 为该层网络的输入矢量，函数返回内积矩阵 Z。函数的调用形式②返回 dotprod 函数导函数的名称，即'ddotprod'。

例 3.72 设定权值矩阵 W 和输入矢量 P，计算内积加权输入。

```
W = [ 0.9   0.6   0.8; 0.2   0.5   0.7 ];
P = [ 0.4   0   0.8 ]';
Z = dotprod ( W, P )
Z =
        1.0000
        0.6400
```

网络属性设置：

newp 和 newlin 函数建立的网络都采用 dotprod 函数作为加权函数。如果要使网络第 *i* 层神经元权值的加权函数为 dotprod 函数，只要将该层输入权值和网络权值的加权函数 net.inputWeights{i,j}.weightFcn 及 net.layerWeights{i,j}.weightFcn 设置为'dotprod'即可。

参见： sim、ddotprod、dist、negdist 和 normprod 函数。

3. mandist

功能： 曼哈顿距离加权函数。

格式：

① Z = mandist (W, P)

② df = mandist ('deriv')

③ D = mandist (pos)

说明：

mandist 函数用于计算两矢量间的曼哈顿距离，其计算公式为

D = sum (abs (x−y))



上式中，D为矢量x和y之间的曼哈顿距离。函数调用时各输入量及返回值的含义参见函数dist。mandist函数没有导函数。

例 3.73　设定一个权值矩阵 W 和输入矢量 P，计算其曼哈顿距离的加权输入。

```
W = [ 0.9   0.6   0.8; 0.2   0.5   0.7 ];
P = [ 0.4   0   0.8 ]';
Z = mandist ( W, P )
Z =
        1.1000
        0.8000
```

例 3.74　设定三个神经元的位置坐标矢量 pos，计算它们之间的曼哈顿距离。

```
pos = [ 0.4   0.9   0.4; 0.6   0.7   0.9; 0.8   0.2   0.9 ];
D = mandist ( pos )
D =
             0      1.2000      0.4000
        1.2000           0      1.4000
        0.4000      1.4000           0
```

网络属性设置：

如果要使网络第 *i* 层神经元权值的加权函数为 mandist 函数，只要将该层输入权值和网络权值的加权函数 net.inputWeights{i,j}.weightFcn 及 net.layerWeights{i,j}.weightFcn 设置为 'mandist'即可。

newsom 函数建立的网络可以采用 mandist 函数作为距离函数。如果要使网络第 *i* 层的距离函数为 mandist 函数，只要将该层的距离函数 net.layers{i}.distanceFcn 设置为'mandist'即可。

参见： sim、dist 和 linkdist 函数。

4．negdist

功能： 负欧氏距离加权函数。

格式：

① Z = negdist (W, P)

② df = negdist ('deriv')

说明：

negdist 函数用于计算两矢量间的负欧氏距离，其计算原理用 MATLAB 语句表示为

```
D = - sum ( ( x - y ).^2 ).^0.5
```

上式中，D为矢量x和y之间的负欧氏距离。

函数调用时各输入量和返回值的含义参见 dotprod 函数。negdist 函数没有导函数。

例 3.75　设定一个权值矩阵 W 和输入矢量 P，计算其负欧氏距离加权输入。

```
W = [ 0.9   0.6   0.8; 0.2   0.5   0.7 ];
P = [ 0.4   0   0.8 ]';
Z = negdist ( W, P )
```



```
Z =
        -0.7810
        -0.5477
```

网络属性设置：

newc 和 newsom 函数建立的网络都采用 negdist 函数作为加权函数。如果要使网络第 i 层神经元权值的加权函数为 negdist 函数，只要将该层输入权值和网络权值的加权函数 net.inputWeights{i,j}.weightFcn 及 net.layerWeights{i,j}.weightFcn 设置为'negdist'即可。

参见： sim、dotprod 和 dist 函数。

5. normprod

功能： 规则化内积加权函数。

格式：

① Z = normprod (W, P)

② df = normprod ('deriv')

说明：

normprod 函数用于计算两矢量之间的规则化内积，其计算公式用 MATLAB 语句表示为

```
z = w*p / sum(p)
```

其中，w为权值矢量，p为输入矢量，z为规则化内积。

函数调用时各输入量和返回值的含义参见 dotprod 函数。normprod 函数没有导函数。

例 3.76 设定权值矩阵 W 和输入矢量 P，计算其规则化内积的加权输入。

```
W = [ 0.9   0.6   0.8; 0.2   0.5   0.7 ];
P = [ 0.4   0   0.8 ]';
Z = normprod ( W, P )
Z =
        0.8333
        0.5333
```

网络属性设置：

newgrnn 函数建立的网络采用 normprod 函数作为加权函数。如果要使网络第 i 层神经元权值的加权函数为 normprod 函数，只要将该层输入权值和网络权值的加权函数 net.inputWeights{i,j}.weightFcn 及 net.layerWeights{i,j}.weightFcn 设置为'normprod'即可。

参见： sim、dotprod、dist 和 negdist 函数。

3.2.14 加权函数的导函数

加权函数的导函数如表 3-15 所示。在加权函数中，只有 dotprod 函数具有导函数。



表 3-15　加权函数的导函数

函 数 名 称	功　　能
Ddotprod	内积加权函数的导函数

ddotprod

功能： 内积加权函数的导函数。

格式：

① dZ_dP = ddotprod ('p', W, P, Z)

② dZ_dW = ddotprod ('w', W, P, Z)

说明：

ddotprod 函数用于计算内积函数的输出对输入矢量的导数。

在函数的调用形式①中，W 为权值矩阵，P 为该层网络的输入矢量，Z 为内积矩阵，函数返回内积函数输出量对输入矢量的导数。函数的调用形式②返回内积函数输出量对权值矩阵的导数。

例 3.77　设定权值矩阵 W 和输入矢量 P，计算内积加权输入及其导数。

```
W = [ 0.9   0.6   0.8; 0.2   0.5   0.7 ];
P = [ 0.4   0   0.8 ]';
Z = dotprod ( W, P )
Z =
dZ_dP = ddotprod ( 'p', W, P, Z )
dZ_dP =
              0.9000      0.6000      0.8000
              0.2000      0.5000      0.7000
dZ_dW = ddotprod ( 'w',W, P, Z )
dZ_dW =
              0.4000
                   0
              0.8000
```

参见：dotprod 函数。

3.2.15　输入函数

输入函数如表 3-16 所示，这类函数确定了神经网络每层的加权输入量以何种方式和神经元阈值组合在一起形成神经元传递函数的输入。

网络属性设置：

如果要设置网络第 i 层神经元的输入函数，只要将该层的输入函数 net.layers{i}.netInputFcn 设置为期望的输入函数即可。



表 3-16 输入函数

函数名称	功能
Netprod	乘积输入函数
Netsum	求和输入函数

1. netprod

功能：乘积输入函数。

格式：

① N = netprod (Z1, Z2, …, Zn)

② df = netprod ('deriv')

说明：

netprod 函数以乘积方式把加权输入和阈值组合在一起。newpnn 和 newgrnn 函数产生的网络以 netprod 函数作为输入函数。

在函数的调用形式①中，Z1 到 Zn 分别为加权输入矩阵或阈值矩阵，此时函数把 Z1 到 Zn 各矩阵中同一位置上的元素分别相乘，然后返回一个维数相同的输入矩阵。函数的调用形式②返回 netprod 函数导函数的名称，即'dnetprod'。

例 3.78 设定加权输入矩阵 Z1、Z2 和阈值矢量 b，计算其乘积输入。

```
Z1 = [ 1 3; 3   4 ];
Z2 = [ -1 2; -5 1 ];
b = [ 0; -1 ];
```

为处理批处理数据，需要把阈值矢量扩展为维数相同的矩阵：

```
B = concur ( b, 2 )
B =
        0        0
       -1       -1
N = netprod ( Z1, Z2, B )
N =
        0        0
       15       -4
```

参见：sim、dnetprod、netsum 和 concur 函数。

2. netsum

功能：求和输入函数。

格式：

① N = netsum (Z1, Z2, …, Zn)

② df = netsum ('deriv')

说明：

netsum 函数以求和方式把加权输入和阈值组合在一起。newp 和 newlin 等函数产生的

网络以 netsum 函数作为输入函数。

在函数的调用形式①中，Z1 到 Zn 分别为加权输入矩阵或阈值矩阵，此时函数把 Z1 到 Zn 各矩阵中同一位置上的元素分别相加，然后返回一个维数相同的输入矩阵。函数的调用形式②返回 netsum 函数导函数的名称，即'dnetsum'。

例 3.79 设定加权输入矩阵 Z1、Z2 和阈值矢量 b，计算其求和输入。

```
Z1 = [ 1 3; 3   4 ];
Z2 = [ -1 2; -5 1 ];
b = [ 0; -1 ];
```

为处理批处理数据，需要把阈值矢量扩展为维数相同的矩阵：

```
B = concur ( b, 2 );
N = netsum ( Z1, Z2, B )
N =
        0       5
       -3       4
```

参见：sim、dnetsum、netprod 和 concur 函数。

3.2.16 输入函数的导函数

输入函数的导函数如表 3-17 所示。

表 3-17 输入函数的导函数

函数名称	功能
Dnetprod	乘积输入函数的导函数
Dnetsum	求和输入函数的导函数

1. dnetprod

功能：乘积输入函数的导函数。

格式：

dN_dZ = dnetprod (Z, N)

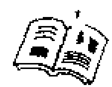
说明：

dnetprod 函数用来求取乘积输入函数的输出对加权输入或阈值的导数。

在调用函数时，Z 为加权输入矩阵或阈值矩阵，N 为乘积输入函数的输出，导函数返回输入函数的输出 N 对加权输入或阈值 Z 的导数矩阵。

例 3.80 设定加权输入矩阵 Z1、Z2 和阈值矢量 b，计算乘积输入值及其对 Z1 的导数。

```
Z1 = [ 1 3; 3   4   ];
Z2 = [ -1 2; -5 1 ];
b = [ 0; -1 ];
B = concur ( b, 2 );
```



```
N = netprod ( Z1, Z2, B );
dN_dZ1 = dnetprod ( Z1, N )
dN_dZ1 =
              0     0
              5    -1
```

参见：netprod、netsum 和 dnetsum 函数。

2. dnetsum

功能：求和输入函数的导函数。

格式：

dN_dZ = dnetsum (Z, N)

说明：

dnetsum 函数用来计算求和输入函数的输出 N 对加权输入或阈值 Z 的导数。函数调用时各输入量和返回值的含义参见 dnetprod 函数。

例 3.81 设定加权输入矩阵 Z1、Z2 和阈值矢量 b，计算求和输入值及其对 Z1 的导数。

```
Z1 = [ 1 3; 3   4 ];
Z2 = [ -1 2; -5 1 ];
b = [ 0; -1 ];
B = concur ( b, 2 );
N = netsum ( Z1, Z2, B );
dN_dZ1 = dnetsum ( Z1, N )
dN_dZ1 =
              1     1
              1     1
```

参见：netsum、netprod 和 dnetprod 函数。

3.2.17 拓扑函数

自组织映射网络的拓扑函数如表 3-18 所示。

表 3-18 拓 扑 函 数

函数名称	功　　能
gridtop	长方形网格拓扑函数
hextop	六边形网格拓扑函数
randtop	任意形状网格拓扑函数

网络属性设置：

如果要设置网络第 i 层的拓扑函数，只要把该层的拓扑函数 net.layers{i}.topologyFcn 设为期望的拓扑函数即可。



1. gridtop

功能：长方形网格拓扑函数。

格式：

pos = gridtop (dim1, dim2, …, dimN)

说明：

gridtop 函数将自组织映射网络的神经元排列在 N 维长方形网格上。函数输入 dim1 到 dimN 是网络在各维上分布的神经元个数，函数返回各神经元的坐标矩阵。

例 3.82 将 16 个神经元排列在 4×4 的长方形网格上，并显示神经元的拓扑结构。

```
pos = gridtop ( 4, 4 );
plotsom ( pos )
```

其拓扑结构如图3.28所示。

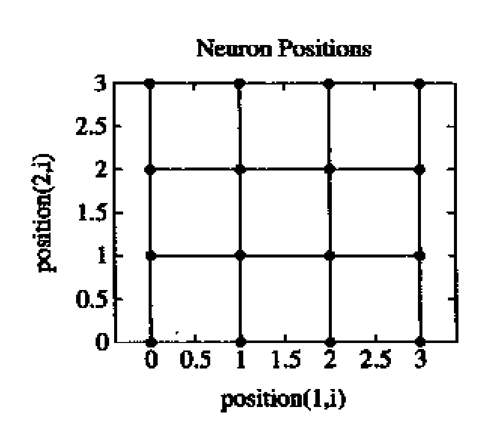


图 3.28　神经元在长方形网格上排列的拓扑图

参见：hextop 和 randtop 函数。

2. hextop

功能：六边形网格拓扑函数。

格式：

pos = hextop (dim1, dim2, …, dimN)

说明：

hextop 函数将自组织映射网络的神经元排列在 N 维六边形网格上。函数输入 dim1 到 dimN 是网络在各维上分布的神经元个数，函数返回神经元的坐标矩阵。

例 3.83 将 16 个神经元排列在 4×4 的六边形网格上，并显示神经元的拓扑结构。

```
pos = hextop ( 4, 4 );
plotsom ( pos )
```

其拓扑结构如图3.29所示。

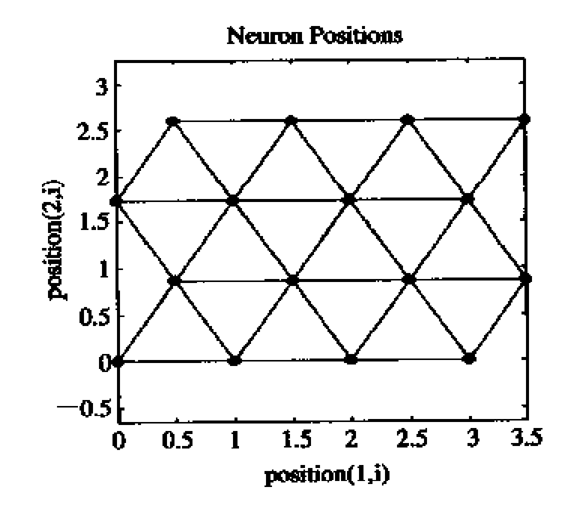


图 3.29　神经元在六边形网格上排列的拓扑图

参见：gridtop 和 randtop 函数。



3. randtop

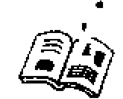
功能：任意形状网格拓扑函数。

格式：

pos = randtop (dim1, dim2, …, dimN)

说明：

randtop 函数将自组织映射网络的神经元排列在 N 维任意形状的网格上。函数输入 dim1 到 dimN 是网络在各维上分布的神经元个数，函数返回神经元的坐标矩阵。



例 3.84 将 16 个神经元排列在 4×4 的任意形状网格上，并显示神经元的拓扑结构。

```
pos = randtop ( 4, 4 );
plotsom ( pos )
```

其拓扑结构如图3.30所示。

参见：gridtop 和 hextop 函数。

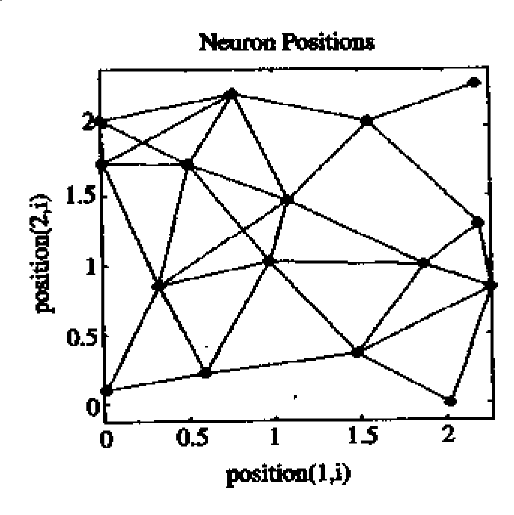


图 3.30 神经元在任意形状网格上排列的拓扑图

3.2.18 距离函数

自组织映射网络的距离函数如表 3-19 所示。

网络属性设置：

如果要设置网络第 *i* 层的距离函数，只要把该层的距离函数 net.layers{i}.distanceFcn 设为期望的距离函数即可。

表 3-19 距 离 函 数

函数名称	功 能
boxdist	Box 距离函数
dist	欧氏距离加权函数
linkdist	Link 距离函数
mandist	曼哈顿距离加权函数

1. boxdist

功能：Box 距离函数。

格式：

D = boxdist (pos)

说明：

该函数用于计算两矢量间的 Box 距离，其计算方式用 MATLAB 语句表示如下：



D = max (abs (x - y))

上式中，D为矢量 x 和 y 之间的距离。当神经元空间分布的拓扑结构由gridtop函数确定时，常采用boxdist函数计算神经元之间的距离。

本函数的输入 pos 为神经元的坐标矩阵，函数返回距离矩阵 D。

例 3.85 设定三个神经元的位置坐标矢量 pos，并计算它们之间的 Box 距离。

```
pos = [ 0.4   0.9   0.4; 0.6   0.7   0.9; 0.8   0.2   0.9 ];
D = boxdist ( pos )
D =
          0    0.6000    0.3000
     0.6000         0    0.7000
     0.3000    0.7000         0
```

参见：sim、mandist 和 linkdist 函数。

2. dist

参见 3.2.13 小节中关于 dist 函数的说明。

3. linkdist

功能：Link 距离函数。

格式：

D = linkdist (pos)

说明：

该函数用于计算两矢量间的 Link 距离，其计算原理用 MATLAB 语句表示如下：

$D_{ij} = 0$ 如果 $i = j$。

$D_{ij} = 1$ 如果 ((sum ((Pi-Pj) .^ 2)) .^ 0.5) <= 1。

$D_{ij} = 2$ 如果存在 k ，使 $D_{ik} = D_{kj} = 1$ 成立。

$D_{ij} = 3$ 如果存在 k_1 、 k_2 ，使 $D_{ik_1} = D_{k_1k_2} = D_{k_2j} = 1$ 成立。

$D_{ij} = M$ 如果存在 k_1 、k_2 、 … 、k_M ， 使 $D_{ik_1} = D_{k_1k_2} = \cdots = D_{k_Mj} = 1$ 成立。

$D_{ij} = S$ 其他

上式中 Pi 和 Pj 分别为某层网络中任意两神经元 *i* 和 *j* 的坐标矢量，D_{ij}是它们之间的距离。

本函数的输入 pos 为神经元的坐标矩阵，函数返回距离矩阵 D。

例 3.86 设定三个神经元的位置坐标矢量 pos，计算它们之间的 Link 距离。

```
pos = [ 1   1   1; 1   2   1; 2   2   1 ];
D = linkdist ( pos )
D =
     0     1     1
     1     0     2
     1     2     0
```

参见：sim、mandist 和 dist 函数。





4. mandist

参见 3.2.13 小节中有关 mandist 函数的说明。

3.2.19 数据预处理和后处理函数

网络训练前、后数据的预处理和后处理函数如表 3-20 所示。

表 3-20 数据预处理和后处理函数

函数名称	功能
Premnmx	把数据归一化到-1 和 1 之间
Postmnmx	恢复被函数 premnmx 归一化的数据
postreg	利用线性回归分析对神经网络的仿真结果进行后处理
prestd	把数据归一化为单位方差和零均值
poststd	恢复被函数 prestd 归一化的数据
prepca	对输入数据进行主元分析
tramnmx	利用预先计算的最大和最小值对数据进行变换
trastd	利用预先计算的均值和方差对数据进行变换
trapca	利用预先计算的主元分析矩阵对数据进行变换

1. premnmx

功能：把数据归一化到-1 和 1 之间。

格式：

① [Pn, minp, maxp, Tn, mint, maxt] = premnmx (P, T)

② [Pn, minp, maxp] = premnmx (P)

说明：

premnmx 函数用于对网络的输入数据或目标数据进行归一化，归一化后的数据将分布在[-1, 1]区间内。归一化公式为

Pn = 2 * (P-minp) / (maxp-minp) - 1

Tn = 2 * (T-mint) / (maxt-mint) -1

其中，P为原始输入数据，maxp和minp分别是P中的最大值和最小值，Pn为归一化后的输入数据。T是原始目标数据，maxt和mint分别是T中的最大值和最小值，Tn是归一化后的目标数据。

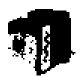
函数的调用形式①可以把网络输入数据 P 和目标数据 T 归一化为 Pn 和 Tn，同时返回 P 中的最小值 minp 和最大值 maxp，以及 T 中的最小值 mint 和最大值 maxt。也可以利用函数的调用形式②只对输入数据 P 进行归一化。

示例参见例 3.87。

参见：prestd、prepca 和 postmnmx 函数。

2. postmnmx

功能：恢复被函数 premnmx 归一化的数据。



格式：

① [P, T] = postmnmx (Pn, minp, maxp, Tn, mint, maxt)

② [P] = postmnmx (Pn, minp, maxp)

说明：

该函数用于恢复被函数 premnmx 归一化的数据，变换公式为

P = 0.5 * (Pn+1) * (maxp−minp) + minp

其中，P为原始数据，maxp和minp分别是P中的最大值和最小值。Pn为归一化后的数据。同理

T=0.5 * (Tn+1) * (maxt−mint) + mint

函数的调用形式①可以在已知 P 中的最小值 minp 和最大值 maxp，以及 T 中的最小值 mint 和最大值 maxt 的前提下，把归一化的网络输入数据 Pn 和目标数据 Tn 恢复为原始数据 P 和 T。也可以利用函数的调用形式②只恢复归一化输入数据 P。

例 3.87 把下列数据归一化到[−1, 1]区间内并恢复其原始数据。

```
p = [-10   -7   -5.5   -2.5   0 ];
t = [   0   7.7   -10   -9.5   0 ];
```

首先对数据进行归一化：

```
[ pn, minp, maxp, tn, mint, maxt ] = premnmx ( p, t )
pn =
         -1.0000    -0.4000    -0.1000     0.5000     1.0000
minp =
         -10
maxp =
           0
tn =
          0.1299     1.0000    -1.0000    -0.9435     0.1299
mint =
         -10
maxt =
          7.7000
```

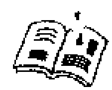
然后恢复原始数据：

```
[ p, t ] = postmnmx ( pn, minp, maxp, tn, mint, maxt )
p =
      -10.0000    -7.0000    -5.5000    -2.5000          0
t =
             0     7.7000   -10.0000    -9.5000          0
```

参见： prestd、prepca 和 premnmx 函数。

3. postreg

功能： 利用线性回归分析对神经网络的仿真结果进行后处理。



格式：

[m, b, r] = postreg (A, T)

说明：

该函数对网络的仿真输出和目标矢量进行线性回归分析，并得到目标矢量对网络输出的相关系数，从而可以作为检验网络性能的参数。

函数的输入分别为网络输出矢量A和目标矢量T，函数返回线性拟合直线的斜率m、截距系数b以及输出矢量A和目标矢量T之间的相关系数r。当r为1时，输出和目标矢量之间的相关性最好。

例 3.88 利用级联前向网络对下列样本数据进行学习，并对网络的实际输出矢量和目标矢量进行线性回归分析。

```
p = [-0.92  0.73  -0.47  0.74  0.29];
net = newff (minmax ( p ), [ 5  1 ],
      { 'tansig'  'purelin' }, 'trainlm' );
t = [-0.08  3.4  -0.82  0.69  3.1];
net = train ( net, p, t );
a = sim ( net, p );
[ m, b, r ] = postreg ( a, t );
```

分析结果如图3.31所示。

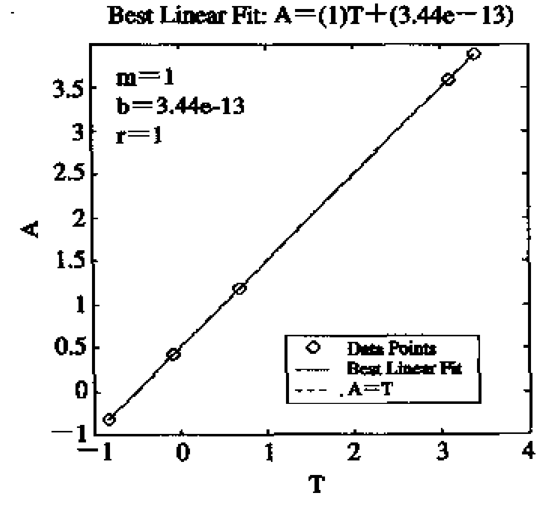


图3.31 线性回归分析结果

参见： premnmx 和 prepca 函数。

4. prestd

功能： 把数据归一化为单位方差和零均值。

格式：

① [Pn, meanp, stdp, Tn, meant, stdt] = prestd (P, T)

② [Pn, meanp, stdp] = prestd (P)

说明：

prestd 函数用于对网络的输入数据或目标数据进行归一化，归一化后的数据将具有零均值和单位方差。归一化公式为

Pn = (P−meanp) / stdp

其中，P和Pn分别为归一化前、后的输入数据，meanp和stdp分别为原始数据P的均值和方差。同理

Tn = (T−meant) / stdt

其中，T和Tn分别是归一化前、后的目标数据，meant是原始数据T的均值，stdt是其方差。

函数的调用形式①可以把网络的输入数据 P 和目标数据 T 归一化为 Pn 和 Tn，同时返回 P 的均值 meanp 和方差 stdp，以及 T 的均值 meant 和方差 stdt。也可以利用函数的调用形式②只对输入数据 P 进行归一化。

示例参见例 3.89。

参见： premnmx、prepca、poststd 和 trastd 函数。



5. poststd

功能：恢复被函数 prestd 归一化的数据。

格式：

① [P, T] = poststd (Pn, meanp, stdp, Tn, meant, stdt)

② [P] = poststd (Pn, meanp, stdp)

说明：

poststd 函数用于恢复被函数 prestd 归一化的数据，变换公式为

P = Pn * stdp + meanp

其中，P和Pn分别为归一化前、后的数据，meanp和stdp分别为原始数据 P 的均值和方差。同理

T = Tn * stdt+meant

函数的调用形式①可以把归一化数据 Pn 和 Tn 恢复为原始数据 P 和 T，同时需要已知 P 的均值 meanp 和方差 stdp，以及 T 的均值 meant 和方差 stdt。也可以利用函数的调用形式②只恢复归一化输入数据 P。

例 3.89　把下列数据归一化为具有零均值和单位方差的数据序列并恢复其原始数据。

```
p = [-10   -7   -5.5   -2.5   0 ];
t = [   0   7.7   -10   -9.5   0 ];
```

首先对数据进行归一化：

```
[ pn, meanp, stdp, tn, meant, stdt ] = prestd ( p, t )
pn =
        -1.2856    -0.5143    -0.1286     0.6428     1.2856
meanp =
        -5
stdp =
        3.8891
tn =
        0.3170     1.3513    -1.0262    -0.9591     0.3170
meant =
        -2.3600
stdt =
        7.4447
```

然后恢复原始数据：

```
[ p, t ] = poststd ( pn, meanp, stdp, tn, meant, stdt )
p =
      -10.0000    -7.0000    -5.5000    -2.5000          0

t =
      0     7.7000   -10.0000    -9.5000          0
```



参见：premnmx、prepca、poststd 和 trastd 函数。

6. prepca

功能：对输入数据进行主元分析。

格式：

[Ptrans, TransMat] = prepca (P, min_frac)

说明：

prepca 函数用来对输入数据矩阵 P 进行主元分析变换，该变换可以消除各输入矢量间的相关性。在变换后的矩阵中，部分矢量占有原始数据矩阵的大部分能量，保存了原始数据的大部分信息，因此变换后应予以保留；而有些矢量只占有原始数据的小部分能量，因此可以在变换后省略。

函数的输入 P 为原始数据矩阵；min_frac 为最小能量比例系数，当变换后矩阵中某一矢量的能量在原始数据总能量中所占的比例小于该系数时，这一矢量将被省略。函数返回经过主元分析的矩阵 Ptrans 和变换时使用的矩阵 TransMat。在该函数算法中都假定原始数据具有零均值，因此要首先利用函数 prestd 对数据进行归一化。

例 3.90　对以下数据进行主元分析：

```
p = [-1.5  -0.58  0.21  -0.96; -2.2  -0.87  0.31  -1.4; -2.5  -0.38  0.44  -1.06 ];
pn = prestd ( p );
[ ptrans, transMat ] = prepca ( pn, 0.02 )
ptrans =
              2.0254    -0.4243    -2.0879    0.4868
transMat =
              -0.5794    -0.5792    -0.5733
```

参见：prestd、premnmx 和 trapca 函数。

7. tramnmx

功能：利用预先计算的最大和最小值对数据进行变换。

格式：

[Pn] = tramnmx (P, minp, maxp)

说明：

本函数利用 premnmx 函数对样本数据进行归一化时得到的最小值和最大值对新的输入数据进行变换。计算公式为

Pn = 2 * (P-minp) / (maxp-minp) - 1

上式中的P和Pn分别为变换前、后的输入数据，maxp和minp分别为premnmx函数找到的最大值和最小值。如果网络训练时所用的是归一化样本数据，那么以后使用网络时所用的新输入数据也应该和样本数据一样接受相同的预处理，这时就要用到tramnmx函数。

例 3.91　利用级联前向网络对下列归一化样本数据进行学习，并用新的输入数据检验网络。



```
p = [-1.92   2.73   -1.47   0.74   2.29 ];
t = [-0.38   3.4    -0.82   0.69   3.1 ];
[ pn, minp, maxp, tn, mint, maxt ] = premnmx ( p, t );
net = newff (minmax ( pn ), [ 5 1 ], { 'tansig'   'purelin' }, 'trainlm' );
net = train ( net, pn, tn );
p2 = [ 2   -2 ];
p2n = tramnmx ( p2, minp, maxp );
a2n = sim ( net, p2n );
a2 = postmnmx (a2n, mint, maxt)
a2 =
        2.9362     -0.2595
```

参见：premnmx、prestd、prepca、trastd 和 trapca 函数。

8. trastd

功能：利用预先计算的均值和方差对数据进行变换。

格式：

[Pn] = trastd (P, meanp, stdp)

说明：

本函数利用 prestd 函数对样本数据进行归一化时得到的均值和方差对新的输入数据进行变换。计算公式为

$$P = Pn * stdp + meanp$$

上式中的P和Pn分别为变换前、后的输入数据，meanp和stdp分别为prestd函数找到的均值和方差。如果网络训练时所用的是归一化样本数据，那么以后使用网络时所用的新输入数据也应该和样本数据一样接受相同的预处理，这时就要用到trastd函数。

例 3.92 利用级联前向网络对下列归一化样本数据进行学习，并用新的输入数据检验网络。

```
p = [-1.92   2.73   -1.47   0.74   2.29 ];
t = [-0.38   3.4    -0.82   0.69   3.1 ];
[ pn, minp, maxp, tn, mint, maxt ] = prestd ( p, t );
net = newff (minmax ( pn ), [ 5 1 ], { 'tansig'   'purelin' }, 'trainlm' );
net = train ( net, pn, tn );
p2 = [ 2   -2 ];
p2n = trastd ( p2, minp, maxp );
a2n = sim ( net, p2n );
a2 = poststd (a2n, mint, maxt)
a2 =
        2.9611     -0.3619
```

参见：premnmx、prestd、prepca、tramnmx 和 trapca 函数。



9. trapca

功能：利用预先计算的主元分析矩阵对数据进行变换。

格式：

[Ptrans] = trapca (P, TransMat)

说明：

本函数利用 prepca 函数对样本数据进行主元分析时得到的变换矩阵 TransMat 对新的输入数据 P 进行变换。如果网络训练时所用的是经过主元分析的样本数据，那么以后使用网络时所用的新输入数据也应该和样本数据一样接受相同的预处理，这时就要用到 trapca 函数。

例 3.93 利用级联前向网络对下列经过主元分析的样本数据进行学习，并用新的输入数据检验网络。

```
p = [-1.92   2.73   -1.47   0.74   2.29;   -2.2   -0.87   0.31   -1.4   -1.2 ];
t =  [-0.38   3.4    -0.82   0.69   3.1 ];
[ pn, meanp, stdp, tn, meant, stdt ] = prestd ( p, t );
[ ptrans, transMat ] = prepca (pn, 0.02 );
net = newff (minmax ( ptrans ), [ 5 1 ], { 'tansig'   'purelin' }, 'trainlm' );
net = train ( net, ptrans, tn );
p2 = [ 2.5   -1.8;   -0.9   -2 ];
p2n = trastd ( p2, meanp, stdp );
p2trans = trapca (p2n, transMat )
a2n = sim ( net, p2trans );
a2 = poststd ( a2n, meant, stdt )
a2 =
          3.1045     -0.9787
```

参见：premnmx、prestd、prepca、trastd 和 tramnmx 函数。

3.2.20 分析函数

分析函数如表 3-21 所示。

表 3-21 分 析 函 数

函 数 名 称	功　　能
errsurf	计算单输入神经元的误差曲面
Maxlinlr	求取线性神经网络的最大学习速率

1. errsurf

功能：计算单输入神经元的误差曲面。

格式：

E = errsurf (P, T, WV, BV, F)



说明：

该函数可用于计算单输入神经元在给定的权值范围矢量和阈值范围矢量情况下的网络误差。函数的输入为神经元输入矢量 P、目标矢量 T、权值范围矢量 WV、阈值范围矢量 BV 以及神经元传递函数 F，函数返回误差曲面各点的误差 E。

例 3.94 神经元的输入和目标矢量分别为

```
p = [-6.1   6   -4.1   -4.0   4.0   4.1   -6   6.1 ];
t = [ 0     1   0.97   0.99   0.01  0.03  0    1 ];
```

给定权值和阈值范围矢量，并计算误差：

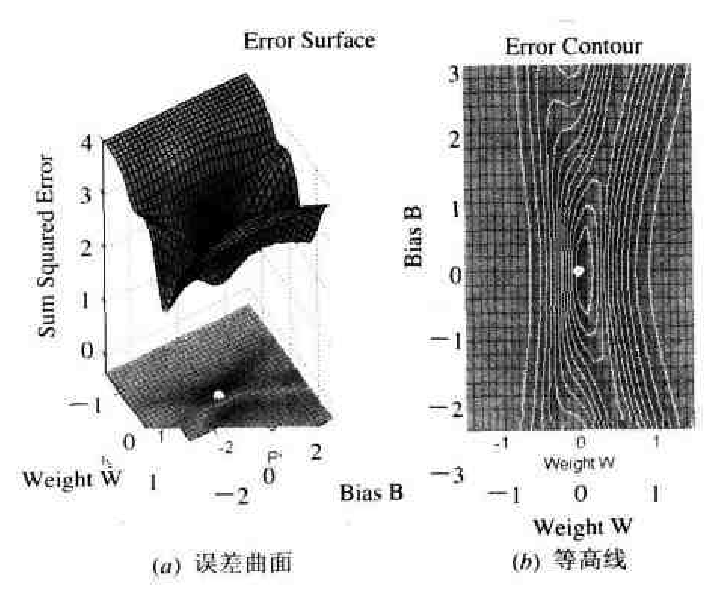
```
wv = -1.5:0.1:1.5;   bv = -3:0.2:3;
es = errsurf (p, t, wv, bv, 'logsig' );
```

绘制误差曲面和等高线，如图 3.32(*a*)、(*b*)所示。

```
plotes (wv, bv, es, [60 30] )
```

设置权值 w、阈值 b 和误差 e，并在误差曲面上标识出这组权值、阈值以及误差点的位置：

```
w = 0;   b = 0;
e = sse (t-logsig ( w * p+b ));
plotep ( w, b, e );
```



(*a*) 误差曲面　　(*b*) 等高线

图 3.32 误差曲面和等高线

其中，图 3.32(*a*)中的亮点对应的误差为 e，图 3.32(*b*)中的亮点对应的权值和阈值分别是 w 和 b。

参见： plotes 和 plotep 函数。

2. maxlinlr

功能： 求取线性神经网络的最大学习速率。

格式：

① lr = maxlinlr (P)





② lr = maxlinlr (P, 'bias')

说明：

maxlinlr函数可以根据线性神经网络的输入矢量P设定网络的学习速率。当网络神经元没有阈值时，使用函数的调用形式①来获得权值的学习速率；当网络神经元有阈值时，使用调用形式②得到权值和阈值的学习速率。

例 3.95 线性神经网络的输入为

P = [4 -4 7; 3 10 6];

对一个有阈值的线性神经网络，根据输入来确定学习速率：

```
lr = maxlinlr ( P, 'bias' )
lr =
        0.0066
```

参见： learnwh 函数。

3.2.21 绘图函数

绘图函数如表 3-22 所示。

表 3-22 绘 图 函 数

函 数 名 称	功 能
hintonw	绘制权值矩阵的 Hinton 图
hintonwb	绘制权值矩阵和阈值矢量的 Hinton 图
plotbr	绘制网络采用 Bayesian 正则化算法训练时的性能变化曲线
plotes	绘制单输入神经元的误差曲面
plotep	在单输入神经元的误差曲面上绘制权值、阈值和相应误差点的位置
plotpv	根据目标矢量绘制感知器的输入矢量
plotpc	在感知器输入矢量图上绘制分类线
plotperf	绘制网络训练过程中的性能变化曲线
plotsom	绘制自组织映射网络图
plotv	绘制起自原点的矢量
plotvec	用不同颜色绘制矢量

1. hintonw

功能： 绘制权值矩阵的 Hinton 图。

格式：

hintonw (W, maxw, minw)

说明：

hintonw 函数以方块图的形式表示权值矩阵，图中各方块的面积正比于权值矩阵中相应元素的大小，方块的颜色表示该元素的符号。其中，红色表示负权值，绿色表示正权



值。该函数的输入为权值矩阵 W、最大权值 maxw 和最小权值 minw。

例 3.96 根据下面的权值矩阵绘制 Hinton 图。

```
W = [ 0.9003   -0.0280   -0.0871    0.1106;
     -0.5377    0.7826   -0.9630    0.2309;
      0.2137    0.5242    0.6428    0.5839 ];
hintonw ( W );
```

运行结果如图3.33所示。

参见：hintonwb 函数。

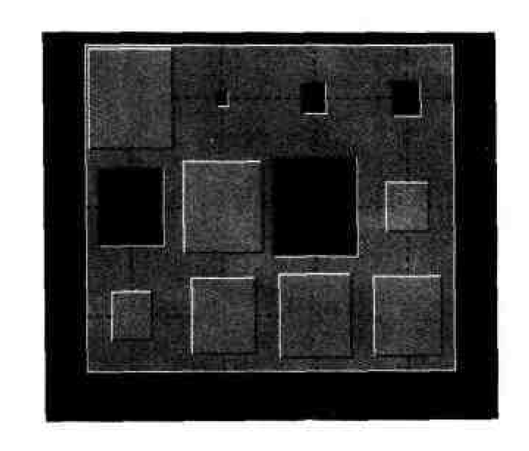


图 3.33 权值的 Hinton 图

2. hintonwb

功能：绘制权值矩阵和阈值矢量的 Hinton 图。

格式：

hintonwb (W, b, maxw, minw)

说明：

hintonwb 函数以方块图的形式表示权值矩阵和阈值矢量，图中各方块的面积正比于权值矩阵或阈值矢量中相应元素的大小，方块的颜色表示该元素的符号，其中，红色表示负值，绿色表示正值。该函数的输入为权值矩阵 W、阈值矢量 b、最大权值 maxw 和最小权值 minw。

例 3.97 根据下面的权值矩阵和阈值矢量绘制 Hinton 图。

```
W = [ 0.9003  -0.0280  -0.0871  -0.1106;
     -0.5377   0.7826  -0.9630   0.2309;
      0.2137   0.5242   0.6428   0.5839 ];
b = [ 0.4503 ;   -0.6124 ;   0.3321 ];
hintonwb ( W, b );
```

运行结果如图3.34所示。

参见：hintonw 函数。

图 3.34 权值和阈值的Hinton图

3. plotbr

功能：绘制网络采用 Bayesian 正则化算法训练时的性能变化曲线。

格式：

plotbr (TR, name, epoch)

说明：

当网络的训练函数为 trainbr 时，可以利用 plotbr 函数绘制训练过程中网络性能的变化曲线，函数绘制的曲线包括网络输出的方差和、各权值参数的平方和以及网络中有效参数的个数。函数的输入 TR 为训练函数产生的训练记录；name 为训练函数名称，缺省值为空字符串；epoch 为要显示的训练次数，其缺省值为训练记录的长度。



例 3.98 假定输入和输出矢量为

```
p = [-1: 0.1: 1 ];
t = sin ( 2*3.14159*p ) + 0.15*randn ( size ( p ) );
```

建立前向网络对上述数据进行学习：

```
net = newff ( [-1 1 ], [ 20, 1 ], { 'tansig', 'purelin' }, 'trainbr' );
[ net, tr ] = train ( net, p, t );
```

在训练过程中，训练函数将自动调用 plotbr 函数绘制性能变化曲线。也可在训练完毕后利用下列语句显示训练记录：

```
plotbr ( tr );
```

网络的性能变化曲线如图 3.35 所示。

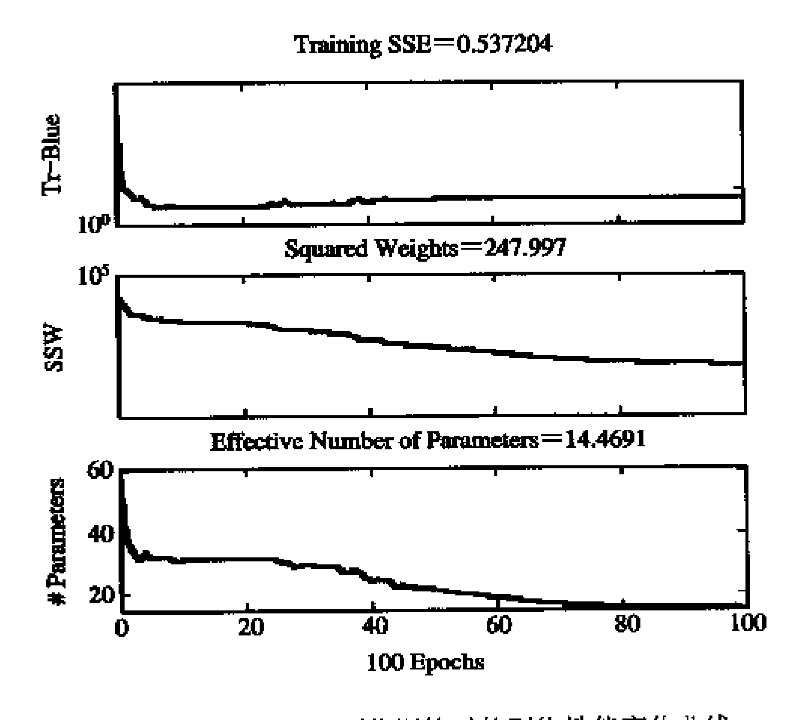


图 3.35 Bayesian 正则化训练时的网络性能变化曲线

4. plotes

功能：绘制单输入神经元的误差曲面。

格式：

plotes (WV, BV, ES, V)

说明：

plotes 函数可用来绘制单输入神经元的误差曲面和等高线。WV 为神经元可能的权值范围矢量；BV 为神经元可能的阈值范围矢量；ES 是由 errsurf 函数计算出的误差矩阵，该矩阵给出了在权值范围矢量和阈值范围矢量各组合点上的神经元输出误差；V 给出了三维图形的视点，缺省值为[-37.5, 30]。

示例参看例 3.94。

参见：errsurf 函数。



5. plotep

功能：在单输入神经元的误差曲面上绘制指定权值、阈值及相应误差点的位置。

格式：

① h = plotep (w, b, e)

② h = plotep (w, b, e, h)

说明：

plotep 函数可以在 plotes 函数所绘的图形上标识出指定权值、阈值和相应误差点的位置。在函数的输入中，w 为权值，b 为阈值，e 为误差，plotep 函数将在等高线图上绘出对应于 w 和 b 的点，在误差曲面上绘出对应于 e 的点，同时函数返回的句柄 h 保存了本次函数所绘各点的信息。调用形式②利用上一次调用函数时返回的句柄 h，在绘制新点前删除旧点。

示例参看例 3.94。

参见：errsurf 和 plotes 函数。

6. plotpv

功能：根据目标矢量绘制感知器的输入矢量。

格式：

① plotpv (P, T)

② plotpv (P, T, v)

说明：

plotpv 函数用于绘制感知器的各输入矢量，图中不同类别的矢量分别用不同的符号标注。函数的输入 P 为感知器输入矢量矩阵，T 是由 0 或 1 组成的二元目标矢量矩阵。当采用调用形式②时，输入 v = [x_min x_max y_min y_max]中的四个元素限定了绘图时坐标轴的显示范围。

示例参看例 3.99。

7. plotpc

功能：在感知器的输入矢量图上绘制分类线。

格式：

① h = plotpc (w, b)

② h = plotpc (w, b, h)

说明：

plotpc 函数将根据给定的感知器权值和阈值在已绘好的感知器输入矢量图上添加分类线。函数的输入 w 和 b 分别是感知器的权值和阈值，函数除绘图外还将返回句柄 h，h 中保存了本次函数所绘分类线的信息。调用形式②利用上一次运行函数时返回的句柄 h，在画新的分类线之前先删除旧的分类线。

例 3.99 根据下列输入矢量和目标矢量绘制某一感知器的输入矢量图：

```
p=[0 0 1 1;0 1 0 1];  t=[1 0 0 0];
plotpv ( p, t ) ;
```



然后利用下面给出的感知器权值和阈值绘出分类线：

```
w = [ 1   1 ];    b = -0.4;
plotpc ( w, b );
```

结果如图 3.36 所示。

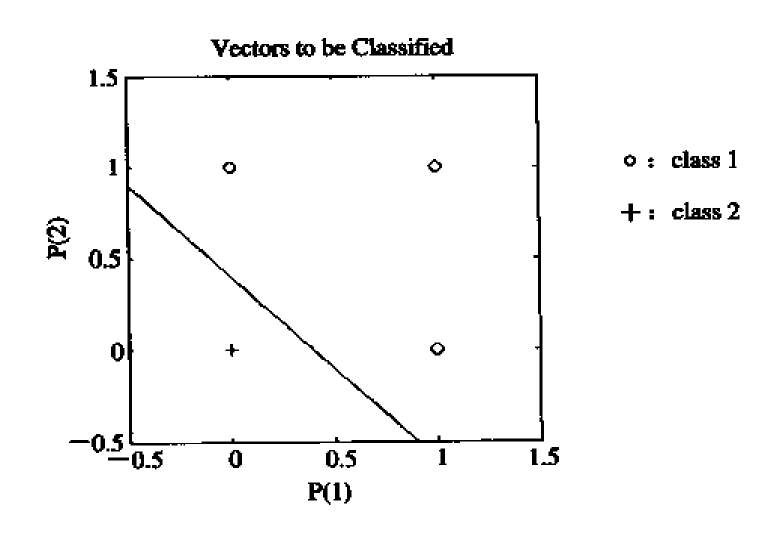


图 3.36 感知器的输入矢量和分类线

参见：plotpv 函数。

8. plotperf

功能：绘制网络训练过程中的性能变化曲线。

格式：

```
plotperf ( TR, goal, name, epoch )
```

说明：

plotperf 函数用于绘制网络在训练时的性能变化曲线，如果训练中有验证步骤和测试步骤，本函数还将绘制验证性能和测试性能的变化曲线。函数的输入 TR 为训练记录；goal 为网络性能目标，缺省值为 NaN；name 为训练函数的名称；epoch 为要显示的训练次数，其缺省值为训练记录的长度。

例 3.100 假定网络的输入、输出和验证数据分别为

```
P = [-1: 0.1: 1 ];
T= sin ( 2*3.14159*P );
VV.P = P;
VV.T = T + rand (1, 21)*0.1;
```

建立前向网络对上述数据进行学习：

```
net = newff (minmax ( P ), [ 4 1 ], { 'tansig', 'tansig' } );
[ net, tr ] = train ( net, P, T, [ ], [ ], VV );
```

在训练过程中，训练函数将自动调用 plotperf 函数绘制网络的性能变化曲线，也可在训练完毕后利用下列语句显示训练记录：



```
plotperf ( tr );
```

网络的性能变化曲线如图3.37所示。

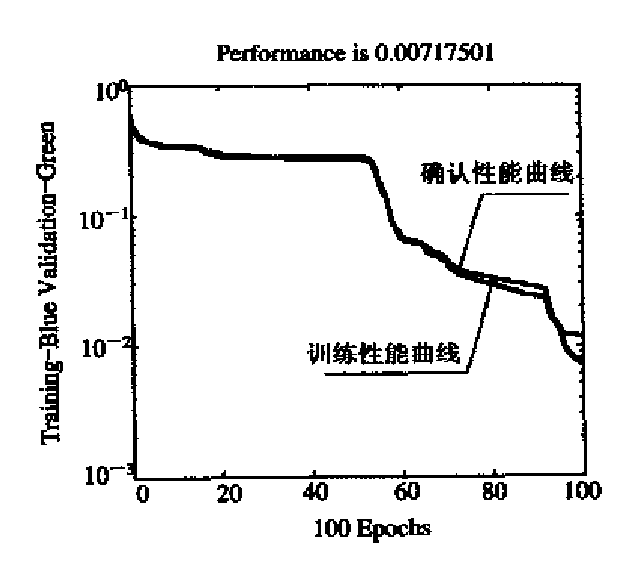


图 3.37　前向网络训练时的性能变化曲线

9. plotsom

功能：绘制自组织映射网络图。

格式：

① plotsom (pos)

② plotsom (W, D, nd)

说明：

plotsom 函数用于绘制自组织映射网络图。在函数的调用形式①中，pos 是网络中各神经元在物理空间分布的位置坐标矩阵；函数返回神经元物理分布的拓扑图，图中每两个间距小于 1 的神经元以直线连接。在函数的调用形式②中，W 为神经元权值矩阵；D 为根据神经元位置坐标计算出的间距矩阵；nd 为邻域半径，缺省值为 1；函数返回神经元权值的分布图，图中每两个间距小于 nd 的神经元以直线连接。

例 3.101　构造一个二输入八神经元的自组织映射网络层，网络的拓扑结构为长方形，神经元间距采用 linkdist 计算方法并随机产生权值矩阵 W。

```
Pos = gridtop ( 2, 4 );
W = rand ( 8, 2 ) ;
```

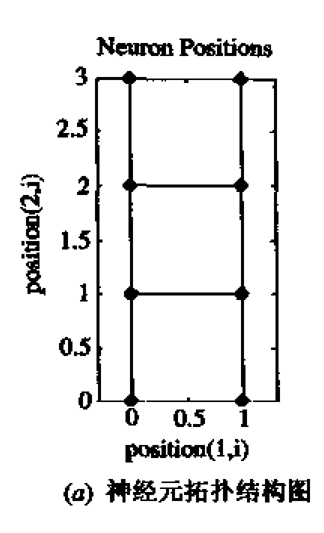
绘制神经元拓扑结构图和权值矢量：

```
subplot ( 1, 2, 1 ), plotsom ( Pos );
D = linkdist ( Pos );
subplot ( 1, 2, 2 ), plotsom ( W, D );
```

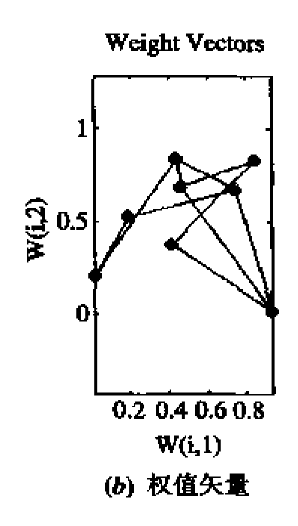
结果如图3.38所示。

参见：newsom 和 learnsom 函数。





(a) 神经元拓扑结构图



(b) 权值矢量

图 3.38 自组织映射网络中的神经元拓扑结构图和权值矢量

10. plotv

功能：绘制起自原点的矢量。

格式：

plotv (M, t)

说明：

plotv 函数用于绘制矢量图，函数的输入 M 为矢量矩阵；t 指定绘图时矢量线的形状，缺省值为'-'；函数将绘制 M 中的每一个列矢量。

示例参看例 3.102。

11. plotvec

功能：用不同颜色绘制矢量。

格式：

plotvec (X, C, m)

说明：

plotvec 函数利用不同颜色绘制矢量。函数的输入 X 为矢量矩阵；C 是表示颜色坐标的行矢量；m 指定绘图时矢量的标识符号。函数将绘制 X 中的每一个列矢量，每个矢量的颜色由矢量 C 中的对应元素确定。

例 3.102 绘制下列矢量：

```
w = [ -0.4   0.7 ; -0.5   0.1 ];
plotv ( w, '-' );
plotvec ( w );
axis ([-0.5   1   -0.6   0.2]);
```

结果如图3.39所示。

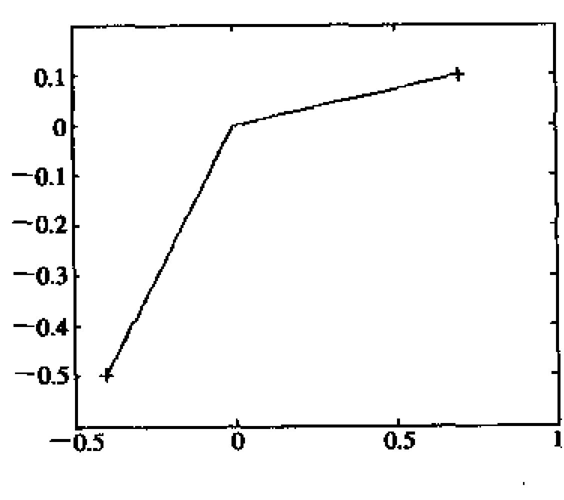


图 3.39 矢量图



3.2.22 网络计算函数

网络计算函数如表 3-23 所示，这类函数用于计算网络的输出、误差等参数，一般由网络的仿真和训练函数调用，而不必由用户调用。

表 3-23 网络计算函数

函数名称	功能
Calca	计算神经网络的输出和其他信号
calca1	计算神经网络一个时间步长的输出
Calce	计算网络各层的误差
calce1	计算网络各层在一个时间步长的层误差
Calcgx	计算网络误差对于权值和阈值的梯度，并用单一矢量表示
calcjejj	计算网络雅可比矩阵的两个相关矢量
calcjx	计算性能函数对于权值和阈值的雅可比矩阵
calcpd	计算网络的延迟输入
calcperf	计算网络的输出和性能
formx	将网络所有的权值和阈值组成一个矢量
getx	以矢量形式获得网络所有的权值和阈值
setx	用一个矢量设置网络所有的权值和阈值

1. calca

功能：计算神经网络的输出和其他信号。

格式：

[Ac, N, LWZ, IWZ, BZ] = calca (net, Pd, Ai, Q, TS)

说明：

calca 函数根据网络输入和各层初始延迟条件计算网络每层的输出。函数的输入 net 为神经网络对象；Pd 为网络的延迟输入；Ai 为各层的初始延迟条件；Q 为批处理数据的个数；TS 为时间步数。函数的返回量 Ac 是由 Ai 和各层输出构成的混合输出，N 为各层神经元传递函数的输入，LWZ 为各层的加权输出，IWZ 为各层的加权输入，BZ 为当前阈值。

2. calca1

功能：计算神经网络一个时间步长的输出。

格式：

[Ac, N, LWZ, IWZ, BZ] = calca1 (net, Pd, Ai, Q)

说明：

calca1 函数根据网络输入和各层初始延迟条件计算网络每层一个时间步长的输出，本函数主要用在 trains 函数中。函数的输入 net 为神经网络对象，Pd 为一个时间步长的延迟输入，Ai 为各层的初始延迟条件，Q 为批处理数据的个数。函数的返回量 A 为该时间步长后网络各层的输出，N 为各层神经元传递函数的输入，LWZ 为各层的加权输出，IWZ 为各层的加权输入，BZ 为当前阈值。





3. calce

功能：计算网络各层的误差。

格式：

El = calce (net, Ac, Tl, TS)

说明：

calce 函数根据网络各层的输出和目标矢量计算每层的误差。net 为神经网络对象，Ac 为各层的混合输出，Tl 为各层的目标矢量，TS 为网络工作的时间步数。函数返回各层的误差 El。

4. calce1

功能：计算网络各层在一个时间步长的层误差。

格式：

El = calce1 (net, Ac, Tl)

说明：

calce1 函数根据网络各层一个时间步长的输出和目标矢量计算每层的误差，本函数主要用在 trains 函数中。本函数的输入 net 为神经网络对象，Ac 和 Tl 分别为各层一个时间步长的混合输出和目标矢量。函数返回各层的误差 El。

5. calcgx

功能：计算网络误差对于权值和阈值的梯度，并用单一矢量表示。

格式：

[gX, normgX] = calcgx (net, X, Pd, BZ, IWZ, LWZ, N, Ac, El, perf, Q, TS)

说明：

calcgx 函数用于计算网络误差对于权值和阈值的梯度。本函数的输入 net 为神经网络对象，X 为权值和阈值组成的矢量，Pd 为延迟输入，BZ 为当前阈值，IWZ 为加权输入，LWZ 为各层加权输出，N 为各层神经元传递函数的输入，Ac 为各层的混合输出，El 为各层的误差，perf 为网络的性能，Q 为当前批处理数据的个数，TS 为网络工作的时间步数；函数返回量 gX 为网络性能对权值和阈值的梯度 dPerf/dX，normgX 为梯度的范数。

6. calcjejj

功能：计算网络雅可比矩阵的两个相关矢量。

格式：

[je, jj, normje] = calcjejj (net, Pd, BZ, IWZ, LWZ, N, Ac, El, Q, TS, MR)

说明：

calcjejj 函数用于计算两个和网络雅可比矩阵有关的矢量，即雅可比矩阵和误差的乘积以及雅可比矩阵的平方，这两个矢量将用来计算网络训练时要用的 Hessian 矩阵。net 为神经网络对象，Pd 为延迟输入，BZ 为当前阈值，IWZ 为加权输入，LWZ 为各层加权输出，N 为各层神经元传递函数的输入，Ac 为各层的混合输出，El 为各层的误差，perf 为网络的性能，Q 为当前批处理数据的个数，TS 为网络工作的时间步数，MR 为内存节省因子。函



数返回量 je 为雅可比矩阵和误差的乘积；jj 为雅可比矩阵的平方，即雅可比矩阵的转置和它本身的内积；normje 为矢量 je 的范数。

7. calcjx

功能：计算性能函数对于权值和阈值的雅可比矩阵。

格式：

jx = calcjx (net, Pd, BZ, IWZ, LWZ, N, Ac, Q, TS)

说明：

calcjx 函数用于计算性能函数对于权值和阈值的雅可比矩阵。函数的输入 net 为神经网络对象，Pd 为延迟输入，BZ 为当前阈值，IWZ 为加权输入，LWZ 为各层加权输出，N 为各层神经元传递函数的输入，Ac 为各层的混合输出，Q 为当前批处理数据的个数，TS 为网络工作的时间步数；函数返回网络的雅可比矩阵 jx。

8. calcpd

功能：计算网络的延迟输入。

格式：

Pd = calcpd (net, TS, Q, Pc)

说明：

calcpd 函数用于计算网络的延迟输入，它是由网络输入和初始输入延迟条件共同确定的。函数的输入 net 为神经网络对象，TS 为网络工作的时间步数，Q 为当前批处理数据的个数，Pc 为混合输入。函数返回网络的延迟输入 Pd。

9. calcperf

功能：计算网络的输出和性能。

格式：

[perf, El, Ac, N, BZ, IWZ, LWZ]= calcperf (net, X, Pd, Tl, Ai, Q, TS)

说明：

calcperf 函数用于计算网络的性能。函数的输入 net 为神经网络对象，X 为权值和阈值组成的矢量，Pd 为延迟输入，Tl 为各层的目标矢量，Ai 为各层的初始输入条件，Q 为当前批处理数据的个数，TS 为网络工作的时间步数；Pc 为混合输入。函数返回量 perf 为网络的性能，El 为各层的误差，Ac 为各层的混合输出，N 为各层神经元传递函数的输入，BZ 为当前阈值，IWZ 为加权输入，LWZ 为各层加权输出。

10. formx

功能：将网络所有的权值和阈值组成一个矢量。

格式：

X = formx (net, B, IW, LW)

说明：

formx 函数将神经网络对象中所有权值参数和阈值参数组合成一个矢量。函数输入 net 为神经网络对象，B 为各阈值参数，IW 为各输入权值参数，LW 为各层的网络权值；函数返回由各权值和阈值构成的矢量 X。



参见：getx 和 setx 函数。

11. getx

功能：以矢量形式获得网络所有的权值和阈值。

格式：

X = getx (net)

说明：

getx 函数用于获得网络的权值和阈值。函数输入神经网络对象 net，返回由各权值和阈值构成的矢量 X。

参见：formx 和 setx 函数。

12. setx

功能：用一个矢量设置网络所有的权值和阈值。

格式：

net = setx (X, net)

说明：

setx 函数用于设置网络的权值和阈值。函数输入为神经网络对象 net 和权值/阈值矢量 X；函数返回已设置好权值和阈值的网络对象。

参见：formx 和 getx 函数。

3.2.23　矢量函数

矢量函数如表 3-24 所示。

表 3-24　矢 量 函 数

函 数 名 称	功　　能
cell2mat	将矩阵构成的单元数组组合成矩阵
combvec	建立矢量所有组合的矩阵
con2seq	将并行矢量转变为串行矢量
concur	复制阈值矢量并构成矩阵
ind2vec	将下标矢量转变为单值矢量组
mat2cell	将矩阵分裂为由子矩阵构成的单元数组
minmax	求矩阵每行的范围
normc	正则化矩阵的列
normr	正则化矩阵的行
pnormc	伪正则化矩阵的列
quant	将实数量化为某一数值的整数倍
seq2con	将串行矢量转变为并行矢量
sumsqr	求矩阵的平方和
vec2ind	将单值矢量组转化为下标矢量

1. cell2mat

功能：将矩阵构成的单元数组组合成矩阵。

格式：

m = cell2mat (c)

说明：

cell2mat 函数可以把矩阵构成的单元数组重组为矩阵。函数的输入 c 是一个单元数组，函数返回由 c 中各元素构成的矩阵。

例 3.103

```
c = { [1] [3]; [4; 7] [8; 9] };
m = cell2mat ( c )
m =
        1      3
        4      8
        7      9
```

参见：mat2cell 函数。

2. combvec

功能：建立矢量所有组合的矩阵。

格式：

a = combvec (a1, a2, …, aN)

说明：

combvec函数将建立输入矩阵a1到aN中各列矢量的所有组合，并返回由这些组合构成的矩阵。

例 3.104

```
a1 = [ 1 2; 4 5 ];
a2 = [ 7; 8 ];
a3 = combvec ( a1, a2 )
a3 =
        1      2
        4      5
        7      7
        8      8
```

3. con2seq

功能：将并行矢量转变为串行矢量。

格式：

① s = con2seq (b)

② s = con2seq (b, TS)



说明：

在神经网络工具箱中，矩阵用于存储神经网络的并行输入矢量，单元数组用于存储网络的串行输入矢量。con2seq 函数可以实现网络并行输入矢量到串行输入矢量的转换。

在函数的调用形式①中，函数输入矩阵 b，返回单元数组 s，s 中的每一个元素为矩阵 b 的列矢量。在函数的调用形式②中，函数输入 b 为 $N\times1$ 维单元数组，b 中的每一个元素是由 $M\times TS$ 列并行输入矢量构成的矩阵，TS 为网络的工作步数；函数返回 $N\times TS$ 维单元数组 s，s 中的每一元素是由 M 列并行输入矢量构成的矩阵。

例 3.105

```
p1 = [ 1 4 2 ];
p2 = con2seq ( p1 )
p2 =
        [1]     [4]     [2]
```

参见：seq2con 和 concur 函数。

4. concur

功能：复制阈值矢量并构成矩阵。

格式：

concur (B, Q)

说明：

concur 函数的输入 B 为阈值矢量，Q 为复制次数，函数返回的矩阵是由 Q 个原阈值矢量构成的矩阵。

例 3.106

```
b = [ 1; 3 ];
concur ( b, 2 )
ans =
        1     1
        3     3
```

参见：seq2con、con2seq、netsum、netprod 和 sim ind2vec 函数。

5. ind2vec

功能：将下标矢量转变为单值矢量组。

格式：

vec = ind2vec (ind)

说明：

ind2vec 函数的输入 ind 是表示类别的下标矢量，函数返回稀疏矩阵 vec，该稀疏矩阵中的每一列只有一个 1，1 的位置由 ind 确定。

例 3.107

```
ind = [ 1 3 2 ];
vec = ind2vec ( ind )
```



```
vec =
        (1,1)        1
        (3,2)        1
        (2,3)        1
```

参见：　vec2ind 函数。

6. mat2cell

功能：将矩阵分裂为由子矩阵构成的单元数组。

格式：

cell = mat2cell (m, r, c)

说明：

mat2cell 函数对输入矩阵 m 进行分裂，分裂后各子矩阵的行数由矢量 r 决定，列数由矢量 c 决定。函数返回由各子矩阵构成的单元数组 cell。

例 3.108

```
M = [ 1 2 3;   5 6 7; 10 11 12 ];
C = mat2cell ( M, [1 2], [2 1] )
C =
        [1x2 double]      [          3]
        [2x2 double]      [2x1 double]
```

参见：　cell2mat 函数。

7. minmax

功能：求矩阵每行的范围。

格式：

pr = minmax (p)

说明：

minmax 函数求取输入矩阵 p 中每一行矢量的取值范围，并返回由各行中最小值和最大值构成的范围矩阵 pr。

例 3.109

```
p = [ 0 1 2 4; -1 -2 -0.5 0 ];
pr = minmax ( p )
pr =
         0      4
        -2      0
```

8. normc

功能：正则化矩阵的列。

格式：

n = normc (m)



说明：

normc 函数对其输入矩阵 m 进行正则化，函数返回的正则化矩阵 n 与原始矩阵维数相同，矩阵每一列元素的平方和变为 1，但各元素之间的比例不变。

例 3.110

```
m = [1 2; 3 4]
n = normc ( m )
n =
        0.3162      0.4472
        0.9487      0.8944
```

参见： normr 函数。

9. normr

功能： 正则化矩阵的行。

格式：

n = normr (m)

说明：

normr 函数对其输入矩阵 m 进行正则化，函数返回的正则化矩阵 n 与原始矩阵维数相同，矩阵每一行元素的平方和变为 1，但各元素之间的比例不变。

例 3.111

```
m = [1 2; 3 4]
n = normr ( m )
n =
        0.4472      0.8944
        0.6000      0.8000
```

参见： normc 函数。

10. pnormc

功能： 伪正则化矩阵的列。

格式：

n = pnormc (m, r)

说明：

pnormc 函数通过对输入矩阵 m 的每一列添加一个元素得到伪正则化矩阵 n，n 中每一列元素的平方和等于输入参数 r 的平方。

例 3.112

```
m = [1 2; 3 4]
n = pnormc ( m, 6 )
n =
        1.0000      2.0000
        3.0000      4.0000
        5.0990      4.0000
```

参见：normc 和 normr 函数。

11. quant

功能：将实数量化为某一数值的整数倍。

格式：

quant (x, q)

说明：

quant 函数将矩阵 x 中的每个元素离散化为量化因子 q 的最近整数倍。

例 3.113

```
x = [ 1.533   4.756   -3.697 ];
y = quant ( x, 0.1 )
y =
        1.5000      4.8000      -3.7000
```

12. seq2con

功能：将串行矢量转变为并行矢量。

格式：

b = seq2con (s)

说明：

在神经网络工具箱中，矩阵用于存储神经网络的并行输入矢量，单元数组用于存储网络的串行输入矢量。seq2con 函数可以实现网络串行输入矢量到并行输入矢量的转换。

函数输入 s 为 $N \times TS$ 维单元数组，s 中的每一个元素是由 M 列矢量构成的矩阵。函数返回 $N \times 1$ 维的单元数组 b，b 中的每一个元素是由 $M \times TS$ 列并行输入矢量构成的矩阵。

例 3.114

```
p1 = { 1 4 2 };
p2 = seq2con ( p1 )
p2 =
          [1x3 double]
```

由上述结果可见，单元数组p2中的惟一元素是一个1×3维矩阵：

```
p2{1}
ans =
          1      4      2
```

参见：con2seq 函数。

13. sumsqr

功能：求矩阵的平方和。

格式：

sumsqr (m)



说明：

sumsqr 函数用来求取函数输入矩阵 m 中各元素的平方和。

例 3.115

```
s = sumsqr ( [ 1 2; 3 4 ] )
s =
        30
```

14. vec2ind

功能： 将单值矢量组转化为下标矢量。

格式：

ind = vec2ind (vec)

说明：

vec2ind 函数的输入 vec 为稀疏矩阵，该矩阵的每列中只有一个 1，1 的位置决定了函数返回矢量 ind 中各元素的值。

例 3.116

```
vec = [ 1   0   0   0;
        0   0   1   0;
        0   1   0   1 ];
ind = vec2ind (vec)
ind =
          1     3     2     3
```

参见： ind2vec 函数。

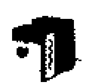
3.3 基于 GUI 的神经网络设计与分析

图形用户界面 GUI（Graphical User Interfaces）是 MATLAB 6.x 神经网络工具箱的新增功能。借助于 GUI，用户可以比直接利用工具箱函数更快捷和方便地完成神经网络的设计与分析。本节我们将在介绍神经网络设计与分析 GUI 的基本功能的基础上，以某一 BP 网络的设计为例给出利用 GUI 进行神经网络设计和分析的基本过程和方法。

3.3.1 神经网络设计 GUI 的基本功能

在 MATLAB 命令窗口键入 nntool 命令即可进入如图 3.40 所示的神经网络设计 GUI 的主界面——网络/数据管理窗口。

图 3.40 所示的 GUI 主界面简洁明了，从中可以清楚地看出该窗口所能实现的主要功能。其中，界面中各列表框分别用于显示神经网络设计中使用或产生的变量和网络对象；Networks and Data 区的各按钮主要用于生成和管理上述变量和网络对象数据；Networks only 区的按钮则主要用于完成神经网络的初始化、仿真和训练。各部分的具体功能如表 3.25 所示。



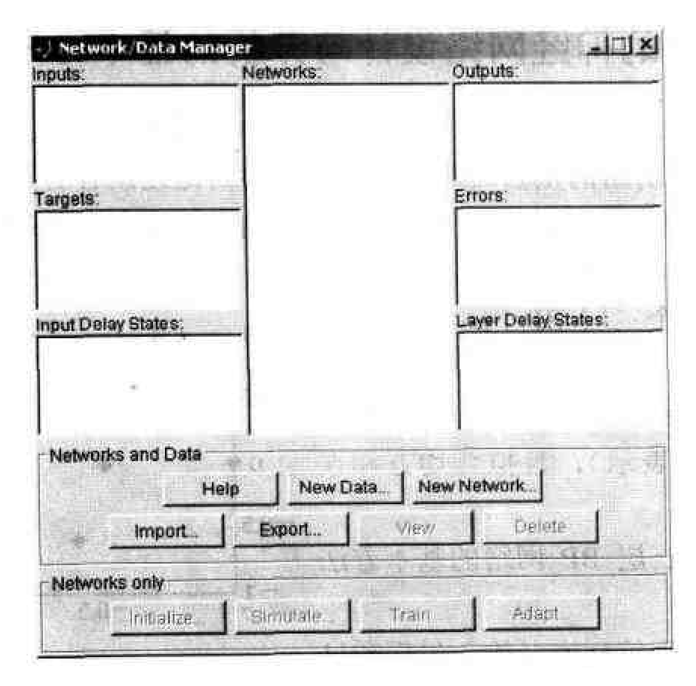


图 3.40　神经网络设计 GUI 主界面

表 3.25　神经网络设计 GUI 主界面的功能列表

项　　目	功　　能
Inputs 列表框	显示神经网络训练所使用的输入数据变量
Targets 列表框	显示神经网络训练所使用的目标输出变量
Outputs 列表框	显示神经网络的实际输出变量
Errors 列表框	显示神经网络的误差变量
Input Delay States 列表框	显示具有输入层延迟神经网络的延迟状态变量
Layer Delay States 列表框	显示具有网络层延迟神经网络的延迟状态变量
Networks 列表框	显示神经网络对象
New Data 按钮	创建新的数据变量
New Network 按钮	创建新的神经网络对象
Import 按钮	从工作空间或文件导入变量或神经网络对象的数据
Export 按钮	将所选变量或神经网络对象的数据导出至工作空间或文件中
View 按钮	查看变量数据或神经网络对象的结构图
Delete 按钮	删除所选变量或神经网络对象
Initialize 按钮	对所选神经网络对象进行初始化
Simulate 按钮	对所选神经网络进行仿真
Train 按钮	对所选神经网络进行批量式训练
Adapt 按钮	对所选神经网络进行渐进式训练
Help 按钮	打开 GUI 帮助文件



3.3.2　基于 GUI 的神经网络设计与分析的基本方法

函数拟合是神经网络的重要应用之一。本节我们将利用 GUI 设计一个 BP 网络，使其能够以较高的精度拟合所给的正弦曲线样本数据，目的是使读者对利用 GUI 进行神经网络设计和分析的基本步骤和方法，以及 GUI 各部分的功能有个大体的了解。对于利用 GUI 进行其他网络的设计过程这里不再赘述。

例 3.117　设计一个三层 BP 网络（输入、输出层神经元数均为 1，中间层神经元数自定），使其能够拟合如图 3.41 所示的正弦曲线样本数据（图中以*号表示），且拟合均方和误差 mse 不超过 0.001。

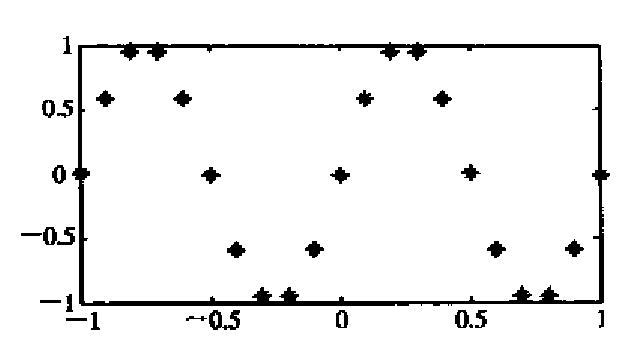


图 3.41　正弦曲线样本数据

利用 GUI 设计该三层 BP 网络的基本方法如下：

(1) 定义神经网络训练样本（输入矢量和目标矢量）数据。

对于函数拟合问题，神经网络的训练样本自然是取待拟合的样本数据，其中，输入矢量取正弦曲线的横坐标样本数据，目标矢量则取相应的纵坐标样本数据。图 3.41 所示的样本数据，可以通过如下语句获得：

输入矢量：p = −1 : 0.1 : 1;

目标矢量：t = sin (2 * pi * p);

(2) 将训练样本数据导入 GUI。

在 GUI 中生成神经网络训练样本数据可以采取两种途径：一种是通过点击 New Data 按钮在 GUI 的弹出窗口中直接输入和定义；另一种方法则是预先在工作空间或文件中生成好所需的数据，然后通过点击 Import 按钮导入 GUI 中。

对于第一种方法，在点击 New Data 按钮后，则弹出如图 3.42 所示的窗口界面。利用该界面读者可以方便地创建新的数据变量，包括变量的名称、内容和类型等。但是，由于在 Value 文本框中只能输入矢量或矩阵数据，显然这对于创建维数和数据量较大的变量来说是不太方便的，对于这种情况，我们可以采用后一种方法来创建数据变量。

对于第二种方法，读者必须首先在工作空间或文件中定义好所需的数据。如前所述，可以在 MATLAB 工作空间中输入如下语句来产生输入矢量和目标矢量：

```
p = -1 : 0.1 : 1;
t = sin ( 2 * pi * p );
```

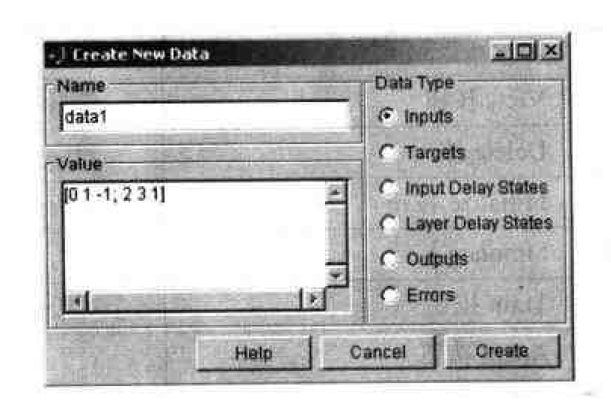


图 3.42　创建新的数据变量的界面

同时，借助于 save 命令可以将生成的数据变量保存至数据文件中：

```
save data p   t;
```

这样，输入矢量 p 和目标矢量 t 均保存至数据



文件 data.mat 中了。

在 GUI 主界面中，点击 Import 按钮，即弹出如图 3.43 所示的界面。

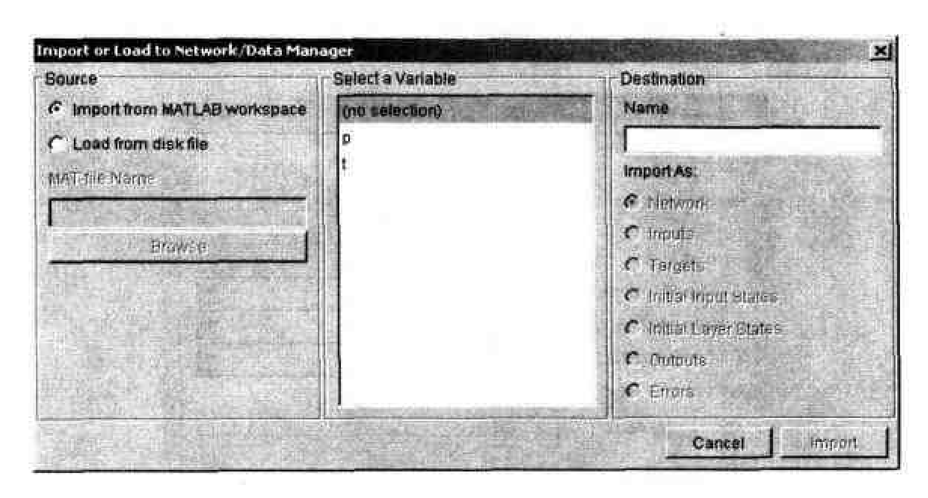


图 3.43 从工作空间或文件导入数据的界面

可见，在 Select a Variable 列表框显示出了当前工作空间中的所有变量，从中可以看到刚才定义的输入矢量 p 和目标矢量 t。依次选择变量 p 和 t，并分别以输入 Inputs 和目标 Targets 类型导入。返回到主界面后，就可以看到导入至 GUI 列表框中的样本数据变量了。在主界面列表框中选择导入变量 p，并点击 View 按钮可以查看导入变量的数据内容，如图 3.44 所示。

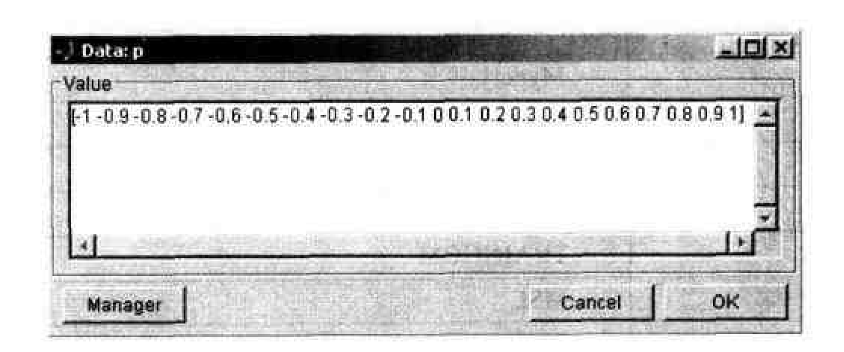


图 3.44 导入变量 p 的数据内容

此外，还可以选择从文件导入数据的方法。在图 3.43 界面中选中 Load from disk file 单选按钮，然后单击 Browse 按钮，系统会弹出如图 3.45 所示的打开文件界面，选择并打开刚才定义好的数据文件 data.mat，同样也可以将样本数据 p 和 t 自动地导入到 GUI 中，在此不再赘述。

(3) 创建神经网络。

在准备好训练样本数据之后，即可着手建立神经网络了。在 GUI 主界面中点击 New Network 按钮，即可进入创建神经网络

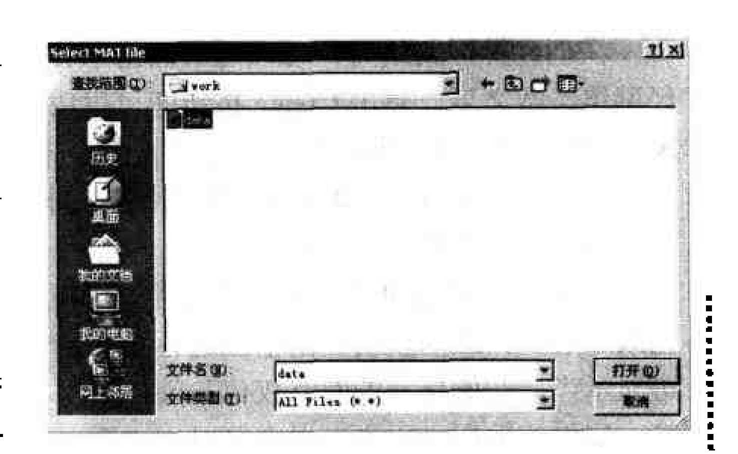


图 3.45 打开 Mat 数据文件的界面



界面，如图 3.46 所示。

在图 3.46 所示界面中可以对神经网络的名称、类型、结构和训练函数等参数进行设置。对于本例，神经网络的各参数设置情况参见表 3.26。

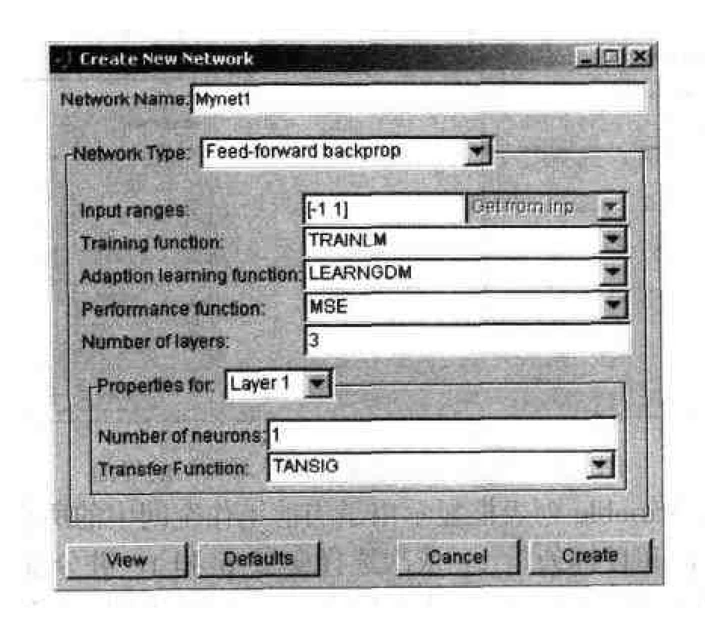


图 3.46 创建神经网络界面

表 3.26 神经网络的参数设置

项　　目	内　　容
网络名称	Mynet1
网络类型	Feed-forward backprop
输入范围	[-1 1]
训练函数	TRAINLM
权值调节规则	TRAINGDM
性能函数	MSE
网络层数	3
各层神经元数目	[1 10 1]
各层传递函数类型	['TANSIG'、'TANSIG'、'PURELIN']

在创建神经网络时，可以随时点击 View 按钮以显示当前设置的网络结构。设置完神经网络参数之后，点击 Create 按钮即可生成相应的神经网络对象。这时，在 GUI 主界面的 Networks 列表框中即可显示当前所创建的神经网络对象。选择 Mynet1 网络对象，点击 View 按钮，可以查看 Mynetl 神经网络的结构示意图，如图 3.47 所示。

(4) 神经网络的初始化。

对于神经网络的初始化，和直接调用工具箱函数生成神经网络一样，在神经网络创建完成之后，网络的权值和阈值已进行了初始化，点击图 3.47 所示界面中的 Weights 页面即可显示当前初始化后的网络权值和阈值数据。图 3.48 中显示的数据即为所创建神经网络的第一层权值数据。用户可以直接对文本框中的数据进行编辑，点击 Set Weight 按钮后则完成了对当前神经网络权值或阈值的修改。



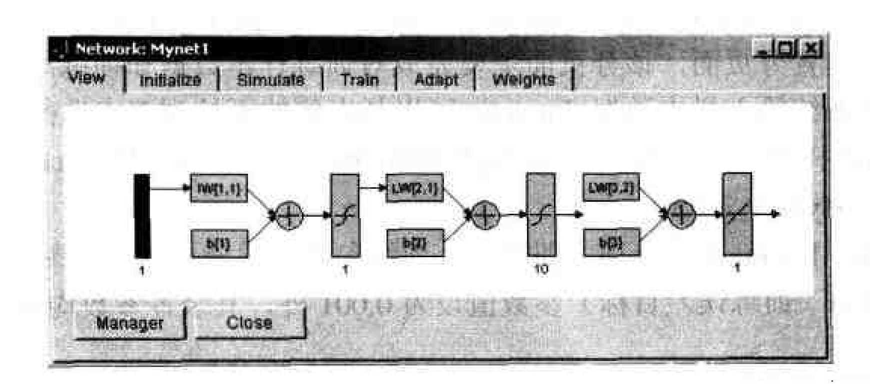


图 3.47　Mynet1 神经网络对象的结构示意图

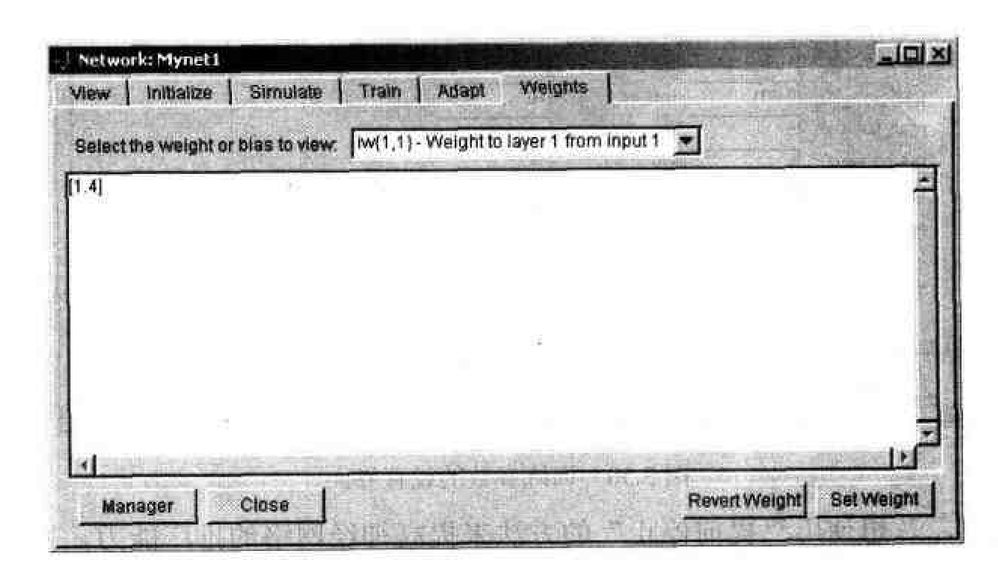


图 3.48　神经网络的权值和阈值

(5) 神经网络的训练。

在主界面中选择创建好的神经网络 Mynet1，然后点击图 3.48 界面中的 Train 项，即可进入神经网络的训练页面，如图 3.49 所示。训练页面又分为三个分页面，包括训练信息

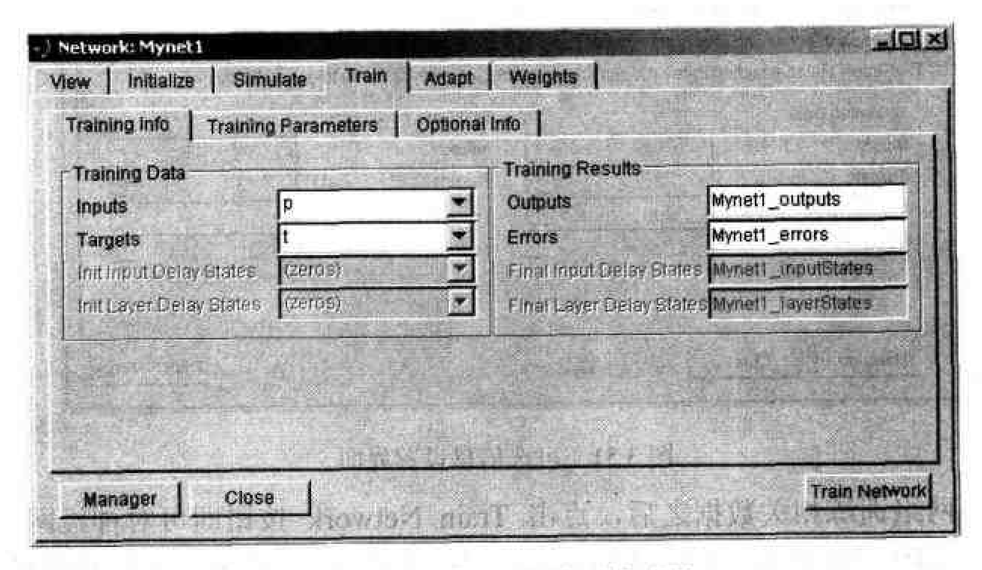


图 3.49　神经网络的训练页面



Training Info、训练参数 Training Parameters 和可选信息 Optional Info 设置页面。图 3.49 显示的为训练信息设置分页面，该分页面主要完成对训练样本数据和训练结果数据信息的设置，其中，对于训练输入样本数据 Inputs，可以从右侧的下拉列表中选择已定义好的变量 p，Targets 项则选择变量 t，将训练结果输出 Outputs 定义为变量 Mynet1_outputs，结果误差 Errors 定义为 Mynet1_errors。

点击 Training Parameters 选项卡即可进入训练参数设置分页面，如图 3.50 所示。在该页面中，除了 goal（训练误差目标）参数值设为 0.001 外，其余各参数均取默认值。

Network: Mynet1

View | Initialize | Simulate | Train | Adapt | Weights

Training Info | Training Parameters | Optional Info

epochs	100	mu_dec	0.1
goal	0.001	mu_inc	10
max_fail	5	mu_max	1000000000(
mem_reduc	1	show	25
min_grad	1e-010	time	Inf
mu	0.001		

Manager | Close | Train Network

图 3.50 训练参数的设置界面

在训练中，若想使用“提前停止”的方法来提高神经网络的推广能力，则可以通过可选信息设置分页面（图 3.51）为网络训练提供所需的验证样本数据和测试样本数据。对于本例网络的训练，此分页面的设置可以省略。

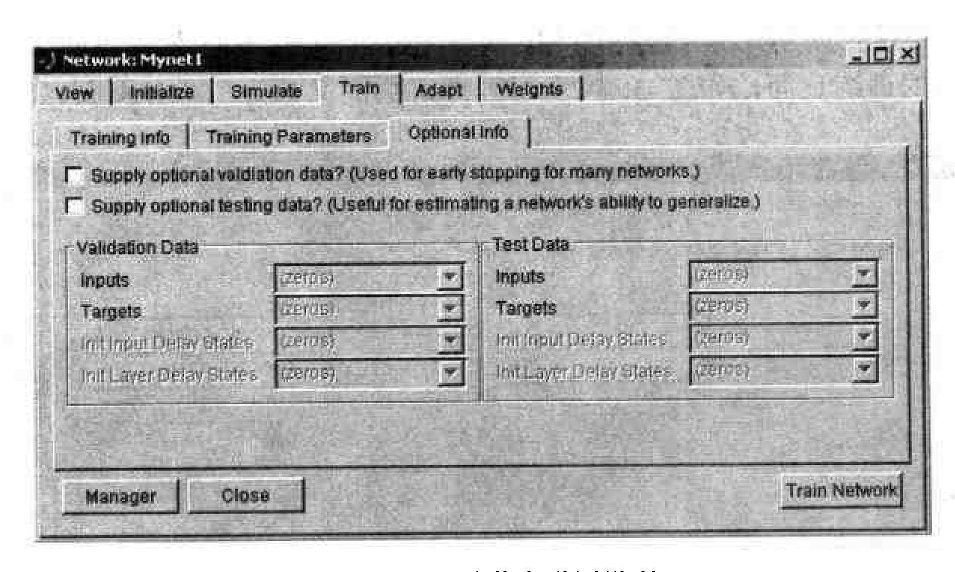


图 3.51 可选信息设置界面

在设置完网络训练相关数据之后，点击 Train Network 按钮即可对神经网络进行训练了，训练结果如图 3.52 所示。由图 3.52 可见，当网络训练至第 20 步时，网络性能达标。



网络训练结束以后，返回 GUI 主界面，可以在 Outputs 和 Errors 列表框中看到刚才定义的训练结果输出变量 Mynet1_outputs 和误差变量 Mynet1_errors 的值。

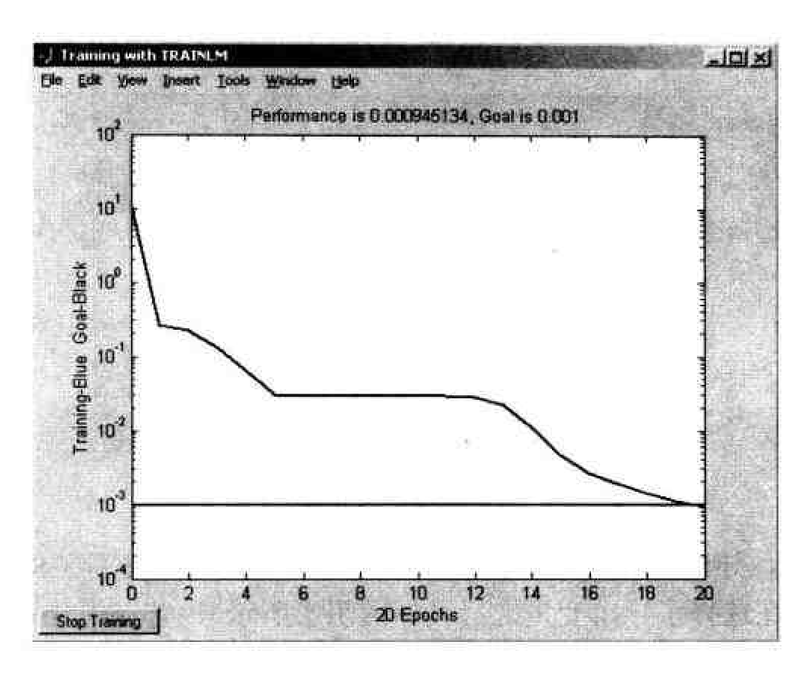


图 3.52 网络训练误差变化曲线

(6) 神经网络的仿真。

图 3.53 所示为神经网络的仿真界面。利用网络输入矢量 p 可以对训练好的网络 Mynet1 进行仿真，仿真结果的输出变量设为 Mynet1_outputs_sim。同时，为了与目标输出进行比较，可以利用所提供的目标矢量 t 来计算仿真结果误差 Mynet1_error_sim。点击 Simulate Network 按钮并返回主界面，此时在 Outputs 和 Errors 列表框中出现了刚才定义的仿真结果变量。

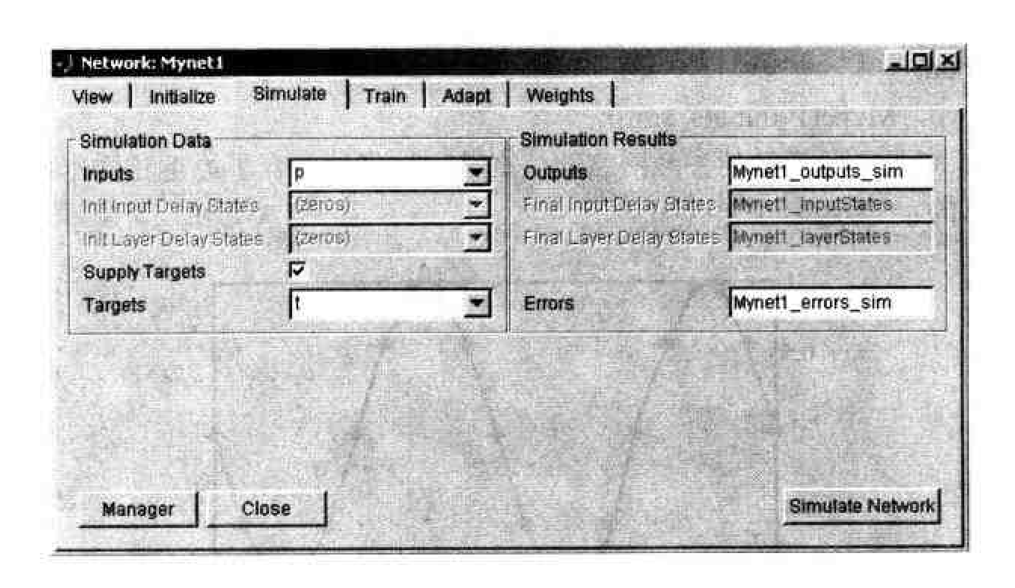


图 3.53 神经网络的仿真页面

(7) 将训练结果和数据导出 GUI。

神经网络设计 GUI 提供了方便的数据导入/导出功能，可以方便地与 MATLAB 工作空



间和磁盘文件进行数据通讯。在 GUI 主界面点击 Export 按钮，则弹出如图 3.54 所示的界面。

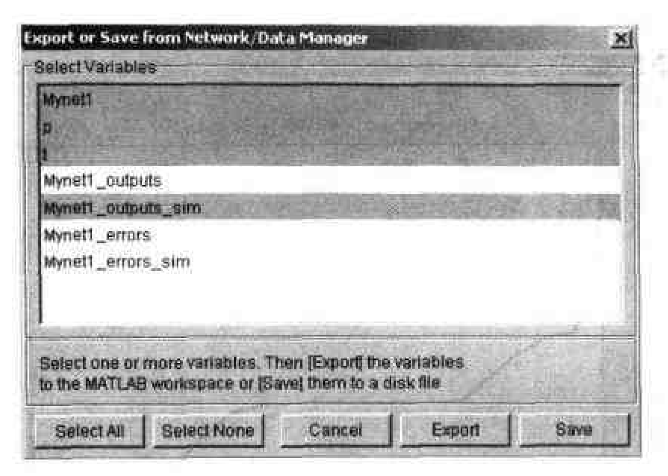


图 3.54 导出数据窗口

由图 3.54 可见，导出数据窗口中显示了当前 GUI 中的所有变量。Select Variables 列表框为一多选列表，在选择了导出变量之后（如图阴影所示），若点击 Export 按钮可以将所选变量导出至工作空间，若点击 Save 按钮则弹出文件保存对话框，可以将所选变量保存至相应的文件中。

点击 Export 按钮，并在工作空间键入 who 命令可以看到导出的数据变量如下：

```
who
Your variables are:
Mynet1          Mynet1_outputs_sim   p       t
```

为了进一步验证神经网络的设计结果，我们可以进一步对仿真结果作如下分析。在命令行键入如下命令，可以得到如图 3.55 所示的拟合曲线。

```
plot ( p, t, '*' );
hold on;
plot ( p, Mynet1_ouputs_sim );
```

由图 3.55 可见，所设计的神经网络的仿真输出较好地拟合了原正弦样本数据，设计结果是十分满意的。

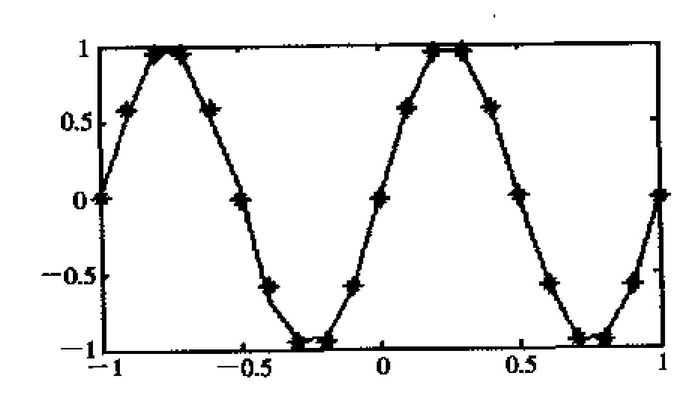


图 3.55 正弦数据的拟合结果



3.4　基于Simulink的神经网络设计与分析

MATLAB 6.x神经网络工具箱提供了专门用于在Simulink中构建和设计神经网络的工具模块，同时，利用gensim函数还可以直接将设计好的神经网络对象转换成Simulink的模块形式，从而可以使用户能够在Simulink中进行神经网络的可视化仿真。此外，MATLAB 6.x神经网络工具箱中还新增了三个专门用于控制系统设计的模块，从而极大地方便了基于神经网络的控制系统设计与仿真。

3.4.1　Simulink神经网络模块介绍

在MATLAB命令行键入命令，

neural

即可调出Simulink神经网络模块组，如图3.56所示。

在图3.56中，Simulink神经网络模块组由四部分组成：传递函数模块Transfer Functions、网络输入模块Net Input Functions、权值单元模块Weight Functions和控制系统模块Control Systems。其中，前三部分是构建神经网络的基本模块，控制系统模块则是专门为构建基于神经网络的控制系统所设计的。下面分别就每一部分的组成及功能作以介绍。

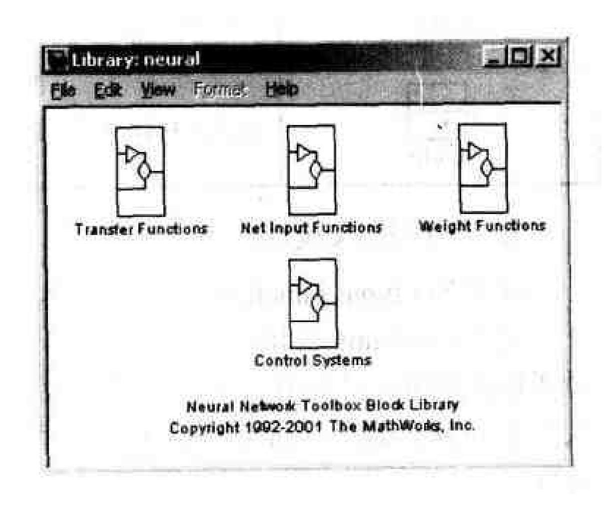


图3.56　Simulink神经网络模块组

1．传递函数模块

双击Transfer Function模块，即可显示如图3.57所示的神经网络传递函数模块。传递函数模块是构建神经网络的基本单元，它们可以接受矢量形式的输入，经过相应的运算，然后以相同维数的矢量形式输出。图3.57中各传递函数模块的类型说明见表3.27。

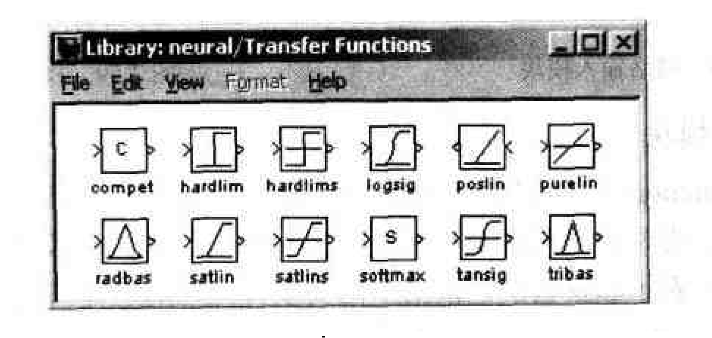


图3.57　神经网络传递函数模块



表 3.27 神经网络传递函数模块的传递函数类型说明

模块形式	传递函数	模块形式	传递函数
C compet	竞争传递函数	radbas	高斯径向基传递函数
hardlim	硬限幅传递函数	satlin	饱和线性传递函数
hardlims	对称硬限幅传递函数	satlins	对称饱和线性传递函数
logsig	对数 sigmoid 传递函数	S softmax	softmax 传递函数
poslin	正线性传递函数	tansig	正切 Sigmoid 传递函数
purelin	纯线性传递函数	tribas	三角基传递函数

2. 网络输入模块

双击 Net Input Functions 模块，进入如图 3.58 所示的网络输入模块组。

其中，netsum 模块可以完成矢量或矩阵的加、减运算，netprod 模块则可用来完成矢量或矩阵的乘、除运算。在神经网络设计中，netsum 模块通常用于接受神经元的加权输入矢量和阈值输入矢量，并将两者相加的计算结果送给传递函数模块以进行下一步的运算。

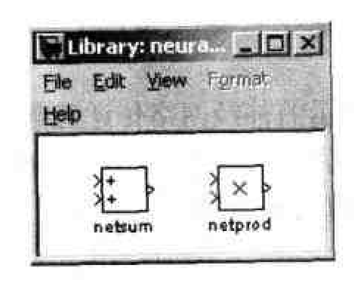


图 3.58 网络输入模块

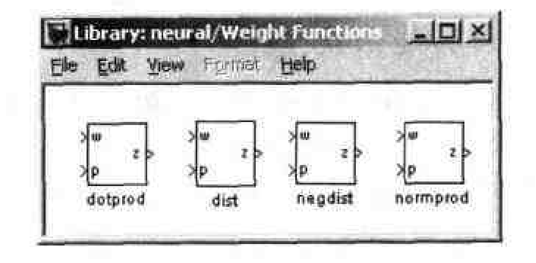


图 3.59 权值单元模块

3. 权值单元模块

双击 Weight Functions 模块则显示如图 3.59 所示的权值单元模块。

权值单元模块主要对神经元的输入进行加权运算，其中 p 表示神经元输入矢量，w 表示网络权值矢量，z 表示加权输出。根据不同类型神经元的运算需要，该模块组共提供了四种加权形式的模块，四种权值单元模块对应的加权运算形式见表 3.28 所示。

表 3.28 权值单元模块的加权运算形式

模块形式	加权运算形式	运算原理
dotprod	点积相乘	z = sum (w .* p)
dist	欧氏距离	z = sum ((w−p) . ^ 2) . ^ 0.5
negdist	欧氏距离取负	z = −sum ((w−p) . ^ 2) . ^ 0.5
normprod	标准化点积相乘	z = w * p / sum (p)

4. 控制系统模块

Simulink 神经网络工具箱为基于神经网络的控制系统设计提供了专用的仿真和设计模块。点击 Control Systems 模块则进入如图 3.60 所示的控制系统模块组。控制系统模块组主要包含了三种神经网络控制系统的设计模块：模型参考控制器模块 Model Reference Controller、反馈线性化控制器模块 NARMA-L2 Controller 和神经网络预测控制器模块 NN Predictive Controller，此外还包含了一个绘图模块 Graph。

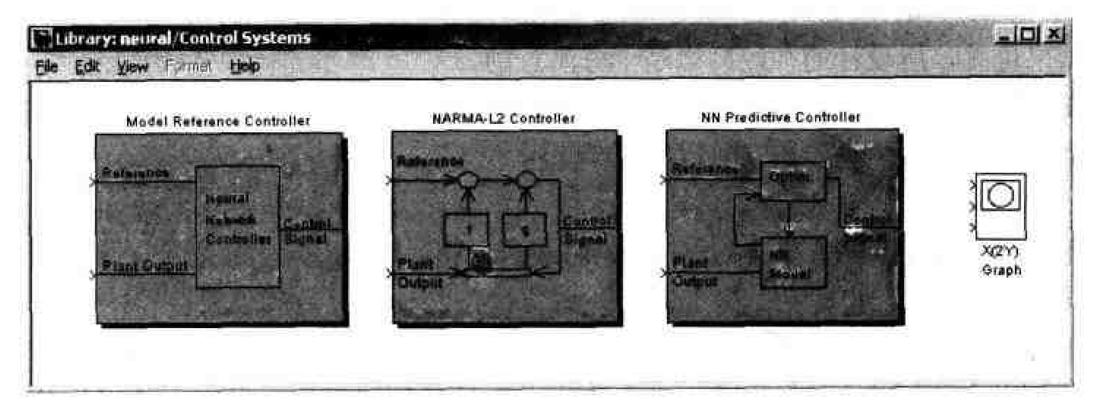


图 3.60 神经网络控制器设计模块组

由于上述三种神经网络控制系统的设计均涉及较深的专业知识，所以这里不再对这三种控制模块的使用作深入介绍。本书 4.7 节给出了一个利用 NN Predictive Controller 模块设计预测控制器的应用实例，以供有兴趣的读者参考。

3.4.2 将神经网络对象转换成 Simulink 形式

利用 gensim 函数可以将设计好的神经网络对象转换成相应的 Simulink 形式，从而可以在 Simulink 中进行神经网络的可视化仿真。gensim 函数的调用格式如下：

```
gensim(net, st )
```

输入项 net 表示神经网络对象，st 表示仿真步长，缺省值为 1。如果当前神经网络对象



中不含有输入或网络延迟环节，则可将 st 设为 -1，表示对神经网络进行连续仿真。

例 3.118 定义输入矢量和目标矢量如下：

输入矢量：p = [-0.5，-0.5，0.3，0；-0.5，0.5，-0.5，1]

目标矢量：t = [2，3，-2，-3]

解：首先，我们利用神经网络工具箱函数设计一个 BP 神经网络，使其能够较好地匹配上面定义的输入 / 输出样本矢量数据。在 MATLAB 工作空间键入如下命令：

```
p = [-0.5, -0.5, 0.3, 0; -0.5, 0.5, -0.5, 1];
t = [2, 3, -2, -3];
net = newff(minmax(p), [2, 1], {'tansig', 'purelin'}, 'trainlm');
net.trainParam.goal = 0.001;
net = init(net);
net = train(net, p, t);
```

训练过程中，显示如下信息：

```
TRAINLM, Epoch 0/100, MSE 6.90866/0.001, Gradient 4.66627/1e-010
TRAINLM, Epoch 4/100, MSE 0.000323004/0.001, Gradient 0.0967428/1e-010
TRAINLM, Performance goal met.
```

训练结束后，利用 sim 函数和样本数据对训练好的神经网络进行仿真：

```
y = sim(net, p)
y =
    1.9984    2.9991    -1.9643    -2.9959
```

由仿真结果可见，所设计的神经网络是十分满意的。

现在我们利用 gensim 函数将设计好的神经网络对象转换成 Simulink 形式。在命令行键入如下命令后，则弹出如图 3.61 所示的窗口。

```
gensim(net, -1);
```

图 3.61 中所示的神经网络模块即为转换之后的 Simulink 形式。有兴趣的读者可以对生成的神经网络模块的内部结构作一些探索，或许会得到一些有益的启发。

下面，我们利用图 3.61 所示系统对神经网络进行仿真。在输入模块 Input1 中将输入量 p{1}设置为[0.3；-0.5]，如图 3.62 所示，点击开始仿真按钮进行仿真。图 3.63 给出了示波器中显示的神经网络输出结果曲线，可见神经网络的仿真输出 y{1}与期望值-2 几乎重合。

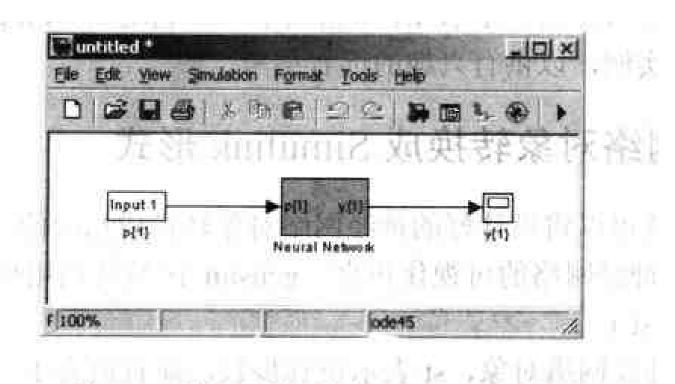


图 3.61 神经网络对象的 Simulink 形式



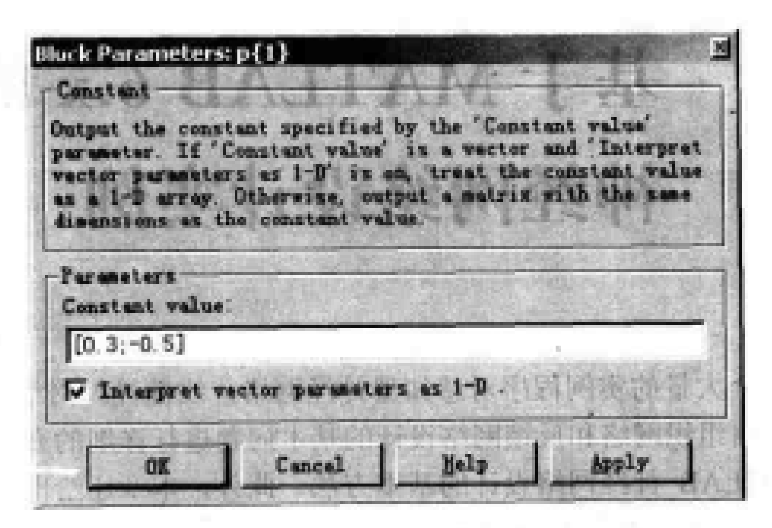


图 3.62　输入参数设置

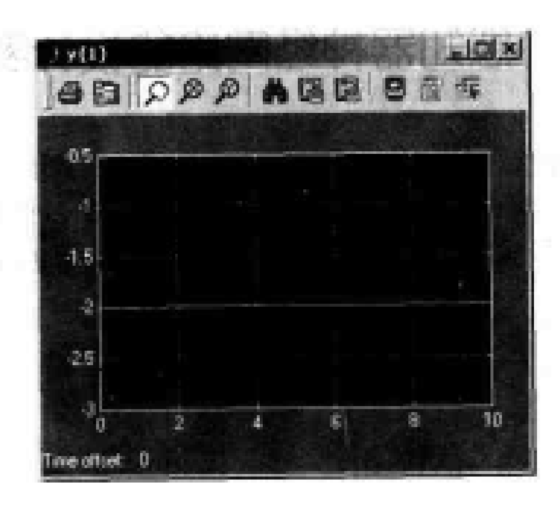
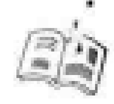
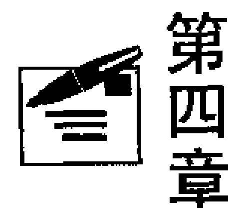


图 3.63　神经网络的仿真结果





第四章　基于 MATLAB 6.x 的神经网络设计实例

本章我们将结合大量的实例程序对感知器神经网络、线性神经网络、BP 神经网络、径向基函数网络、自组织网络和反馈网络设计的基本问题进行详细的介绍，使读者能够熟练地掌握基于 MATLAB 神经网络设计的基本方法。此外，本章还给出了多个应用实例以说明神经网络如何用于解决实际问题。

4.1　感知器神经网络的设计实例

例 4.1　利用感知器神经网络解决一个简单的数据点分类问题：将四个数据点分为两类，一类代号为 1，另一类代号为 0。由于在直角坐标系中，任何数据点均可以用由横坐标和纵坐标组成的二维矢量来表示，所以四个数据点即构成了四个二维输入矢量，即

输入矢量为　$p=\begin{bmatrix}-0.5 & -0.5 & 0.3 & -0.1\\ -0.5 & 0.5 & -0.5 & 1.0\end{bmatrix}$

目标分类矢量为　$t=[1\ \ 1\ \ 0\ \ 0]$

解：首先定义输入矢量及相应的目标矢量：

```
P = [ -0.5 , -0.5 , 0.3 , -0.1 ; -0.5 , 0.5 , -0.5 , 1.0 ];
T = [ 1 , 1 , 0 , 0 ];
```

待分类的数据点可以用图 4.1 来描述，其中，代号为 1 的点用符号“+”表示，代号为 0 的点用符号“o”表示。图 4.1 可以通过函数 plotpv 来绘制，即

```
plotpv ( P , T );
```

根据问题的规模，感知器神经网络采用具有二维输入的单神经元即可。利用函数 newp 生成感知器网络：

```
net = newp ( minmax ( P ) , 1 );
```

利用函数 plotpc 可以画出当前感知器的决策边界：

```
linehandle = plotpc ( net.IW{1} , net.b{1} );
```

由于当前感知器网络的权值和阈值均为零，所以决策边界未能在图 4.1 上画出。下面利用 adapt 函数对感知器网络进行训练，直至网络误差 E 为零。

```
E = 1;
n = 0;
```



```
while ( sse ( E ) )
    [ net , y , E ] = adapt ( net , P , T );
    n = n + 1;
    perf ( n ) = sse ( E );
    % 绘制分类线
    linehandle = plotpc ( net.IW{1} , net.b{1} , linehandle ); drawnow;
end
figure;
plot ( perf );      % 绘制误差变化曲线
```

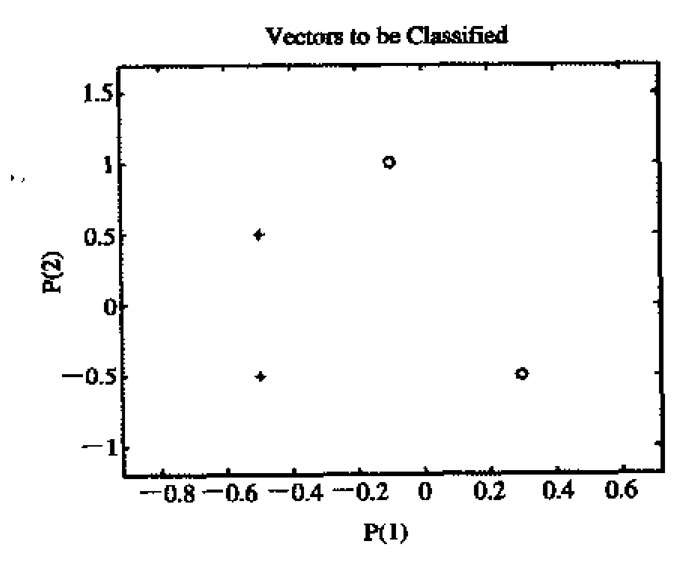


图 4.1 待分类的数据点

网络训练结束后得到如图 4.2 所示的分类结果，可见分类线已将两类数据点分开。图 4.3 给出了相应的误差变化曲线。显然，经过三步训练后，网络训练误差为零。

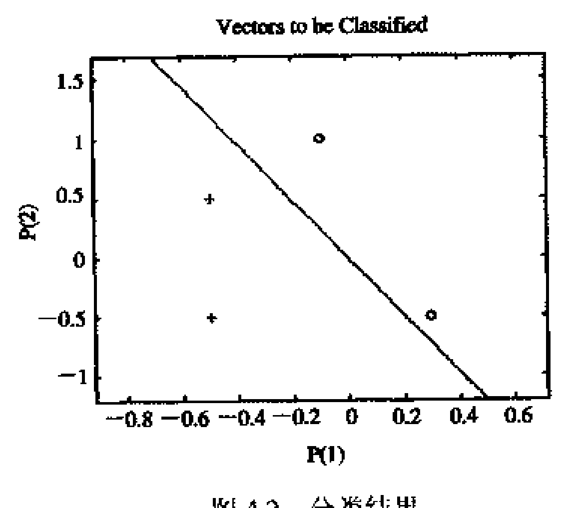


图 4.2 分类结果

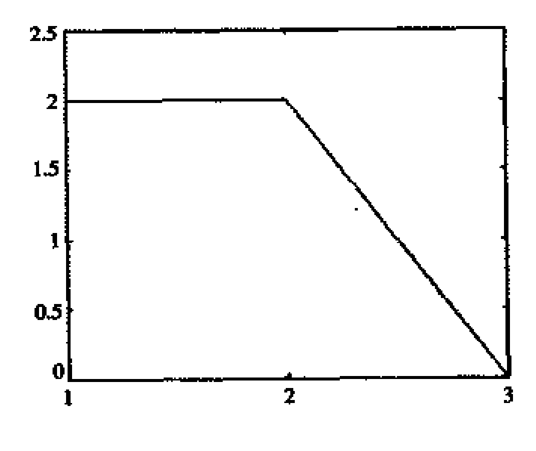


图 4.3 误差变化曲线



经过训练后，网络的权值和阈值分别为

```
W = net.IW{1}
W =
    -1.2000    -0.5000
b = net.b{1}
b =
    0
```

当训练完成之后，即可利用训练好的感知器神经网络来解决实际的分类问题了，这时应利用 sim 函数来实现。

现在引入一个新的数据点[0.7; 1.2]，并利用感知器网络对其进行归类：

```
p = [ 0.7 ; 1.2 ];
a = sim ( net , p )
a =
    0
```

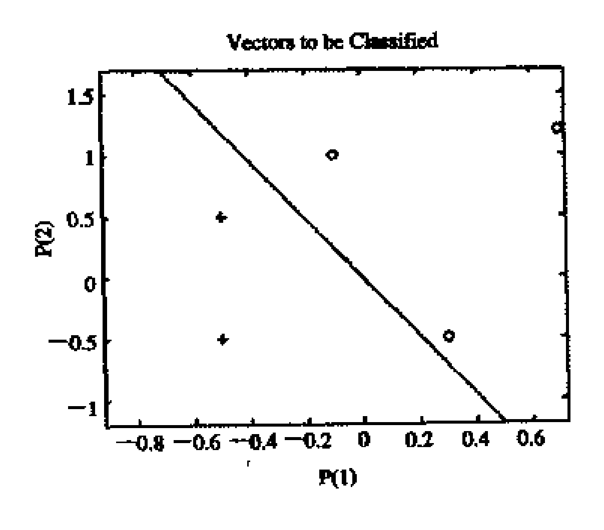


图 4.4　新的分类结果

利用下列语句将新的数据点在图 4.2 中画出，并得到新的分类结果，如图 4.4 所示。

```
plotpv ( p , a );
hold on;
plotpv ( P , T );
plotpc ( net.IW{1} , net.b{1} );
```

下面给出本例完整的 MATLAB 程序：

```
% Example 4.1
%
close all
clear
clf reset
figure(gcf);
echo on
clc
% NEWP——创建感知器神经网络
% ADAPT——对感知器神经网络进行训练
% SIM——对感知器神经网络进行仿真
pause         % 敲任意键开始
clc
% P 为输入矢量
P= [-0.5   -0.5    0.3   -0.1;
     -0.5    0.5   -0.5    1];
```

```
% T 为目标矢量
T=[1 1 0 0];
% 绘出待分类的数据点图
plotpv(P,T);
pause
clc
% 创建感知器神经网络
net=newp([-1 1; -1 1],1);
linehandle=plotpc(net.IW{1},net.b{1});      % 绘制当前决策曲线
pause
clc
echo off
E=1; n=0;
echo on
clc
% 训练感知器神经网络
while(sse(E))
    [net,y,E]=adapt(net,P,T);
    n=n+1;
    perf(n)=sse(E);
    linehandle=plotpc(net.IW{1},net.b{1},linehandle);drawnow;
end
pause
clc
% 绘制误差曲线
plot(perf);     % 绘制误差变化曲线
pause
clc
% 利用训练好的感知器网络进行分类
p=[0.7;1.2];     % 新的数据点
a=sim(net,p)
plotpv(p,a);     % 绘制新的数据点
pause
clc
% 绘制新的分类结果
hold on;
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
echo off
```



例 4.2 复杂输入矢量的分类问题。与例 4.1 不同的是，该问题中待分类的数据点为三维空间的点（如图 4.5），即

$$\text{输入矢量为}\quad p=\begin{bmatrix}-1 & +1 & -1 & +1 & -1 & +1 & -1 & +1\\ -1 & -1 & +1 & +1 & -1 & -1 & +1 & +1\\ -1 & -1 & -1 & -1 & +1 & +1 & +1 & +1\end{bmatrix}$$

$$\text{日标分类矢量为}\quad t=[0\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1]$$

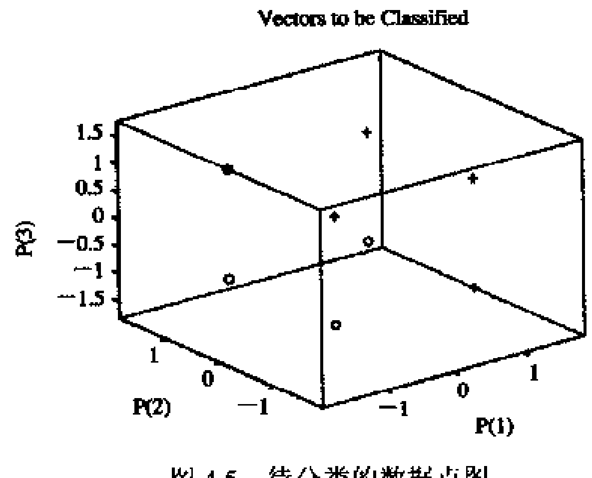


图 4.5 待分类的数据点图

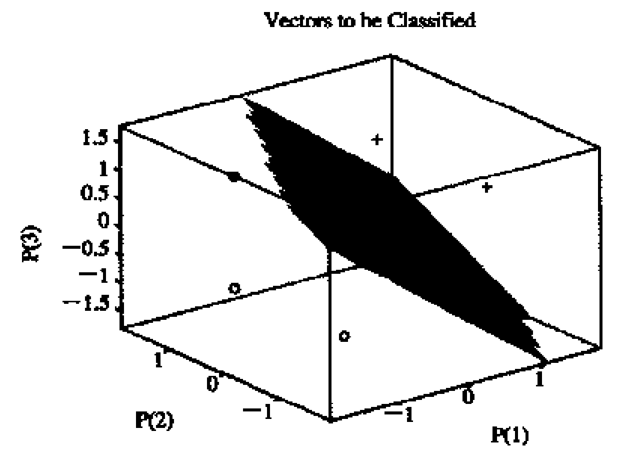


图 4.6 分类结果

解：根据问题的不同，本例中感知器网络结构我们采用输入维数为 3 的感知器神经元。由于输入维数为 3，所以神经网络的决策边界为三维空间的决策平面。图 4.6 和图 4.7 给出了分类结果图及网络训练误差变化曲线，图 4.8 给出了在引入两个新的数据点[0, 1; -0.5, 0.5; 1, 0]后网络的分类结果，可见，感知器网络实现了正确的分类。本例完整的 MATLAB 程序如下：

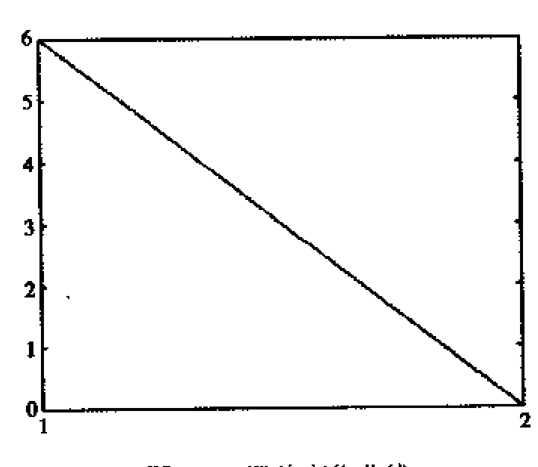


图 4.7 误差变化曲线

图 4.8 新的分类结果

```
% Example 4.2
%
close all
clear
```



```
clf reset
figure(gcf);
echo on
clc
% NEWP——创建感知器神经网络
% ADAPT——对感知器神经网络进行训练
% SIM——对感知器神经网络进行仿真
pause     % 敲任意键开始
clc
% P 为输入矢量
P=[-1,  1, -1,  1, -1,  1, -1, 1;
   -1, -1,  1,  1, -1, -1,  1, 1;
   -1, -1, -1, -1,  1,  1,  1, 1];
% T 为目标矢量
T=[0,1,0,0,1,1,0,1];
% 绘出待分类的数据点图
plotpv(P,T);
pause
clc
% 创建感知器神经网络并初始化
net=newp(minmax(P),1);
net.inputWeights{1,1}.initFcn='rands';
net.biases{1}.initFcn='rands';
net=init(net);
planehandle=plotpc(net.IW{1},net.b{1});     % 绘制当前决策平面
pause
clc
echo off
E=1;
n=0;
echo on
clc
% 训练感知器神经网络
while(sse(E))
    [net,y,E]=adapt(net,P,T);
    n=n+1;
    perf(n)=sse(E);
    planehandle=plotpc(net.IW{1},net.b{1},planehandle);drawnow;
```



```
end
pause
clc
% 绘制误差变化曲线
plot(perf);
pause
clc
% 利用训练好的感知器网络进行分类
p=[0,1;-0.5,0.5;1,0];       % 引入两个新的数据点
a=sim(net,p)
plotpv(p,a);       % 绘制新的数据点
pause
clc
% 绘制新的分类结果
hold on
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
echo off
```

值得注意的是，在神经网络初始化时，若取不同的初始值，感知器神经网络的训练结果会有所不同，但经过训练后的网络仍然能够完成分类任务。在本例程序中，感知器神经网络的权值和阈值在进行初始化时采用了随机初始化函数，所以在每次运行程序后，就会得到不同的分类结果。此外，随着网络初始值的不同，训练所需的步数和误差变化也不相同，但不论何种结果，感知器神经网络最终都圆满地解决了分类问题，这说明该分类问题的解不只一个。

例 4.3 本例我们将说明奇异输入样本矢量对感知器神经网络训练结果的影响。所谓奇异样本，是指相对于其他输入样本特别大或特别小的样本矢量。奇异样本矢量的存在会使基于常规感知器学习规则 learnp 的网络训练效率下降。在例 4.1 的基础上，增加一个新的样本矢量点 [-40;100]，即

输入矢量为 $$p=\begin{bmatrix}-0.5 & -0.5 & 0.3 & -0.1 & -40\\ -0.5 & 0.5 & -0.5 & 1.0 & 100\end{bmatrix}$$

目标分类矢量为 $$t=[1 \quad 1 \quad 0 \quad 0 \quad 1]$$

解：由于样本点 [-40;100] 明显不同于其他输入样本矢量，因此它是奇异样本点。奇异样本点的加入使得感知器网络的训练时间大大加长。下面给出了本例的 MATLAB 程序及相关的分类结果图（如图 4.9～图 4.12 所示），读者可以将本例感知器神经网络的训练效果和例 4.1 作以比较。

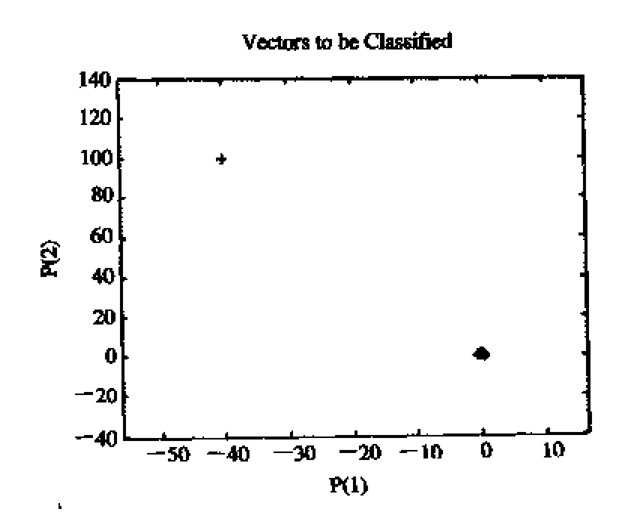


图 4.9　待分类的数据点图

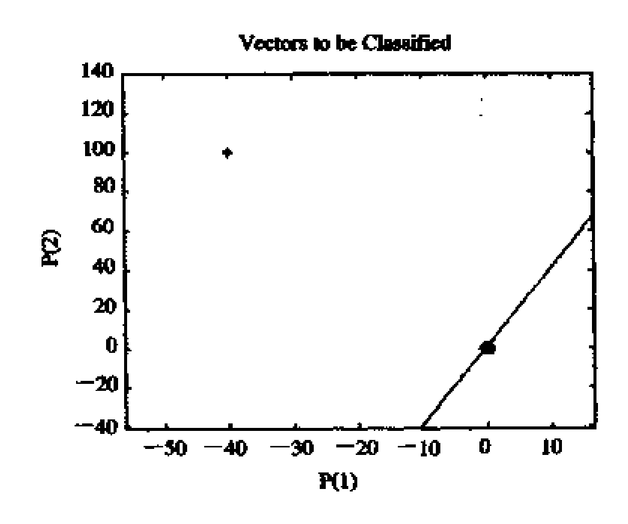


图 4.10　分类结果

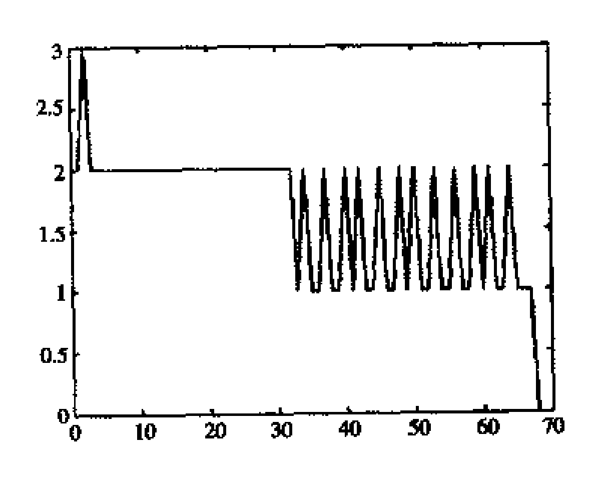


图 4.11　误差变化曲线

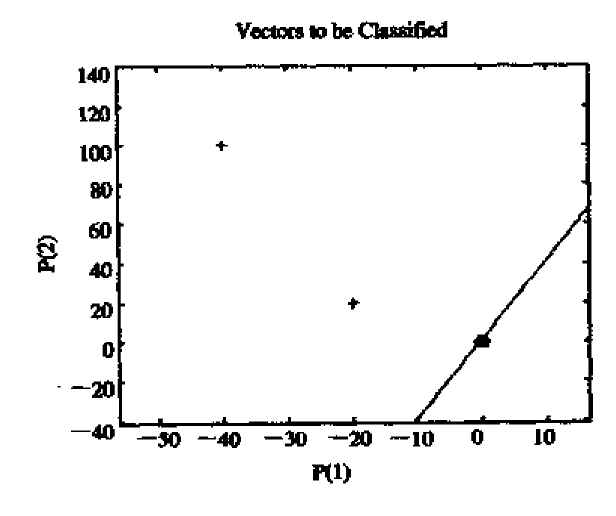
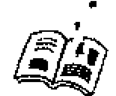


图 4.12　新的分类结果

本例完整的 MATLAB 程序如下：

```
% Example 4.3
%
close all
clear
clf reset
figure(gcf);
echo on
clc
% NEWP——创建感知器神经网络
% ADAPT——对感知器神经网络进行训练
% SIM——对感知器神经网络进行仿真
pause     % 敲任意键开始
```



```
clc
% P 为输入矢量
P=[-0.5, -0.5,  0.3, -0.1, -40;
   -0.5,  0.5, -0.5,  1,   100];
% T 为目标矢量
T=[1,1,0,0,1];
% 绘出待分类的数据点图
plotpv(P,T);
pause
clc
% 创建感知器神经网络
net=newp(minmax(P),1);
linehandle=plotpc(net.IW{1},net.b{1});
pause
clc
echo off
E=1; n=0;
echo on
clc
% 训练感知器神经网络
while(sse(E))
    [net,y,E]=adapt(net,P,T);
    n=n+1;
    perf(n)=sse(E);
    linehandle=plotpc(net.IW{1},net.b{1},linehandle);drawnow;
end
pause
clc
% 绘制误差变化曲线
plot(perf);
pause
clc
% 利用训练好的感知器网络进行分类
p=[-20;20];     % 新的数据点
a=sim(net,p)
plotpv(p,a);     % 绘制新的数据点
pause
clc
% 绘制新的分类结果
```



```
hold on;
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
echo off
```

例 4.4　感知器归一化学习规则的应用。在例 4.3 中，我们发现当输入样本矢量中存在奇异样本时，感知器神经网络的训练时间将大大加长。本例我们将采用第二章 2.2.2 节中所介绍的“感知器归一化学习算法”，即 learnpn 函数对感知器网络进行训练，从而可以消除学习训练时间对奇异样本的敏感性。

解：本例的程序与例 4.3 中的几乎完全相同，只是要把例 4.3 中的感知器网络创建语句

```
net=newp(minmax(P),1);
```

改为如下形式：

```
net=newp(minmax(P),1,'hardlim','learnpn');
```

图 4.13～图 4.16 给出了感知器训练效果和分类结果。读者可以将本例的训练效果和例 4.3 的训练效果作以比较。（本例的 MATLAB 程序略）

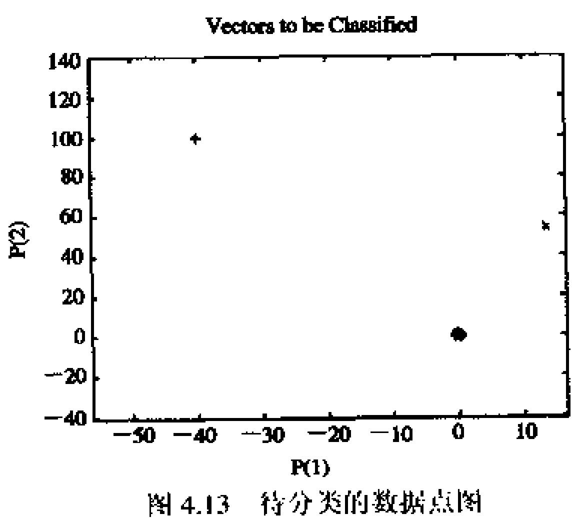


图 4.13　待分类的数据点图

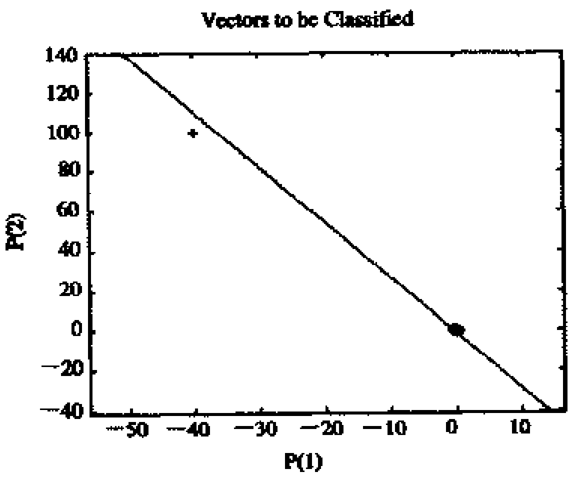


图 4.14　分类结果

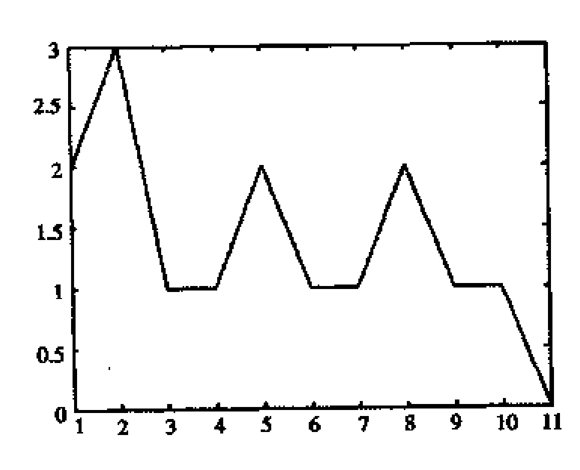


图 4.15　误差变化曲线

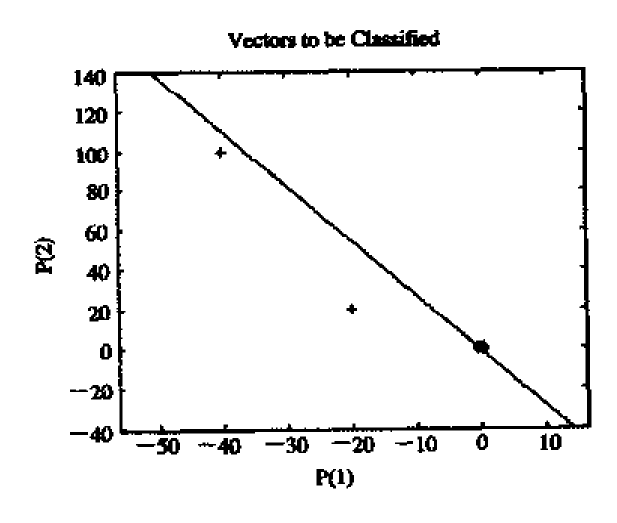


图 4.16　新的分类结果



例 4.5 本例我们将尝试采用单层感知器网络对一组线性不可分的数据点进行分类。通过本例，读者可以更清楚地看到，正如第二章所述，单层感知器网络只能解决线性可分问题，而对于线性不可分输入矢量的分类问题则是无能为力的。定义不可分点集如下：

输入矢量为 $p=\begin{bmatrix}-0.5 & -0.5 & 0.3 & -0.1 & -0.9\\ -0.5 & 0.5 & -0.5 & 1.0 & 0.1\end{bmatrix}$

目标分类矢量设为 $t=[1\ \ 1\ \ 0\ \ 0\ \ 0]$

解：下面给出了本例的 MATLAB 程序和相关的训练结果图。其中，图 4.17 表示待分类的输入矢量集，图 4.18 为感知器网络训练后的分类结果，图 4.19 是训练过程的误差变化曲线。

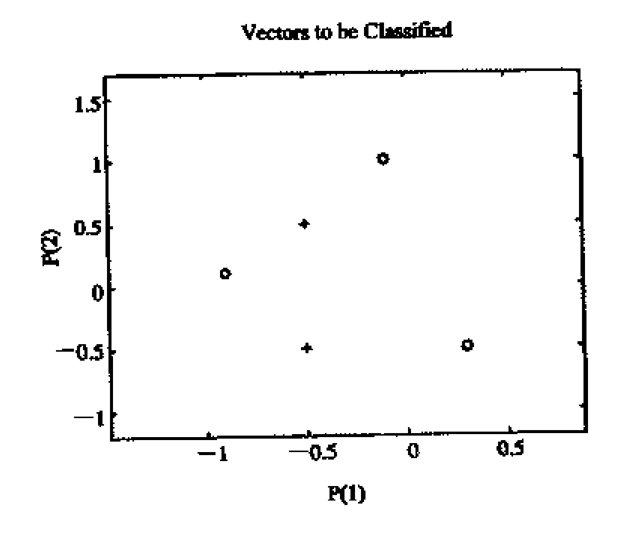


图 4.17 线性不可分点集

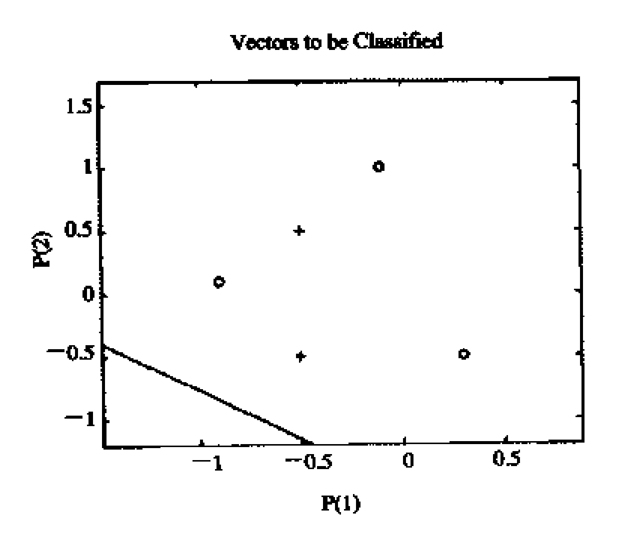


图 4.18 分类结果

```
% Example 4.5
%
close all
clear
clf reset
figure(gcf);
echo on
clc
% NEWP——创建感知器神经网络
% ADAPT——对感知器神经网络
          进行训练
% SIM——对感知器神经网络进行
        仿真
pause      % 敲任意键开始
clc
% P 为输入矢量
```

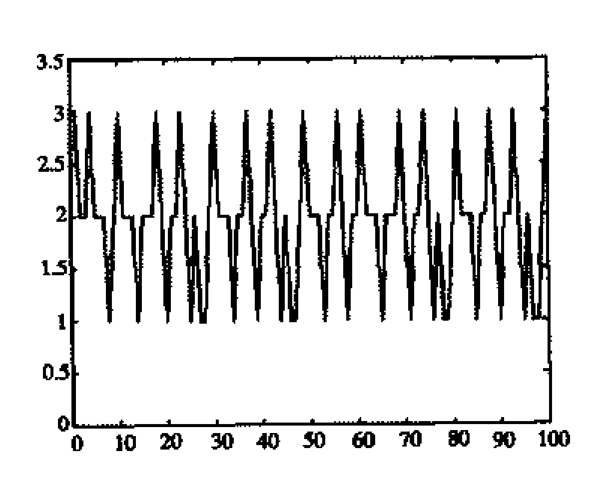


图 4.19 误差变化曲线



```
P=[-0.5, -0.5,  0.3, -0.1, -0.9;
   -0.5,  0.5, -0.5,  1,    0.1];
% T 为目标矢量
T=[1,1,0,0,0];
% 绘出待分类的数据点图
plotpv(P,T);
pause
clc
% 创建感知器神经网络
net=newp(minmax(P),1);
linehandle=plotpc(net.IW{1},net.b{1});
pause
clc
% 训练感知器神经网络
for n=1:30
    [net,y,E]=adapt(net,P,T);
    perf(n)=sse(E);
    linehandle=plotpc(net.IW{1},net.b{1},linehandle);drawnow;
end
pause
clc
% 绘制误差变化曲线

plot(perf);
echo off
```

由图 4.18 和图 4.19 可见，当训练迭代至第 100 步时仍未收敛，而且呈现出有规律的振荡，所以对于本例问题采用单层感知器网络是永远也找不到正确分类方案的。

4.2 线性神经网络的设计实例

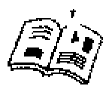
例 4.6 用直接设计法设计一个简单的线性神经元，使其能够拟合如下所给的输入样本矢量和目标矢量，其中

输入矢量为 $p=[0.8 \quad -2]$

目标矢量为 $t=[0.5 \quad 1]$

解：首先，给出本例的 MATLAB 程序如下：

```
% Example 4.6
%
```



```
close all
clear
clf reset
figure(gcf);
echo on
clc
% NEWLIN——采用直接法设计线性神经网络
% SIM——对线性神经网络进行仿真
pause       % 敲任意键开始
clc
P=[0.8,-2];     % P 为输入矢量
T=[0.5,1];      % T 为目标矢量
% 设定线性神经元可能的权值和阈值范围
w_range=-1:0.1:1;
b_range=-1:0.1:1;
% 根据输入和目标矢量绘制线性神经元的误差曲面
ES=errsurf(P,T,w_range,b_range,'purelin');
plotes(w_range,b_range,ES);
pause
clc
% 用直接设计法设计线性神经网络
net=newlind(P,T);
pause
clc
% 对线性神经网络进行仿真
A=sim(net,P)
% 计算仿真误差
E=T-A
SSE=sse(E)
pause
clc
% 根据网络权值和阈值在误差曲面上绘制当前位置点
plotes(w_range,b_range,ES);
plotep(net.IW{1,1},net.b{1},SSE);
echo off
```

在利用 newlind 函数设计线性神经元之前，可以根据可能的网络权值和阈值范围，并利用函数 errsurf 绘制出神经元的误差曲面。本例所绘制的误差曲面如图 4.20 所示，其中越亮的部分误差越小。图 4.20 中所示的亮点为网络设计完成之后根据当前网络权值和阈值，并利用 plotep 函数绘制的误差位置点，可见该误差点位于误差曲面的最低点。



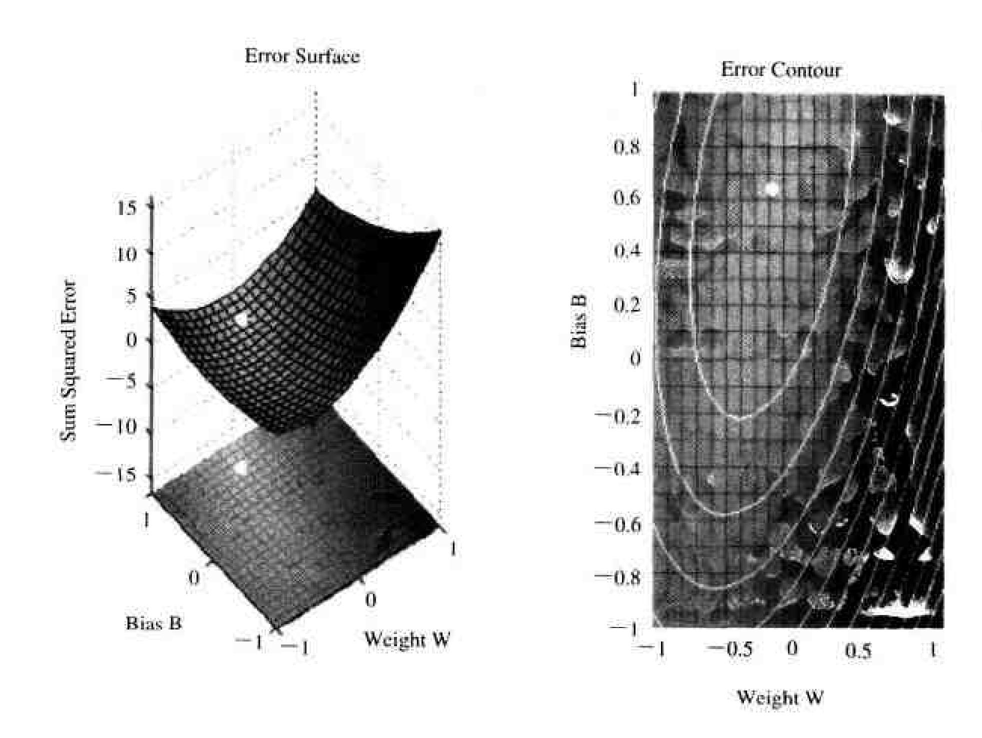


图 4.20　线性神经元的误差曲面和误差点

例 4.7　使用 Train 函数训练并设计一个简单的线性神经元，其网络训练样本矢量和例 4.6 相同。

解： 本例的 MATLAB 程序如下：

```
% Example 4.7
%
close all
clear
clf reset
figure(gcf);
echo on
clc
% NEWLIN——生成一个新的线性神经网络
% TRAIN——对线性神经网络进行训练
% SIM——对线性神经网络进行仿真
pause      % 敲任意键开始
clc
% P 为输入矢量
P=[0.8, -2];
% T 为目标矢量
T=[0.5, 1];
% 设定线性神经元可能的权值和阈值范围
w_range=-1:0.2:1;
```



```
b_range=-1:0.2:1;
% 根据输入和目标矢量绘制线性神经元的误差曲面
ES=errsurf(P,T,w_range,b_range,'purelin');
plotes(w_range,b_range,ES);
pause
clc
% 计算最快的稳定学习速率值
maxlr=maxlinlr(P,'bias')
pause
clc
% 创建线性神经网络
net=newlin(minmax(P),1,[0],0.4*maxlr);
% 在误差曲面上选择权值和阈值对线性神经元进行初始化
plotes(w_range,b_range,ES);
subplot(1,2,2);
[net.IW{1,1},net.b{1}]=ginput(1);
echo off
SSE=sse(T-sim(net,P));
h=plotep(net.IW{1,1},net.b{1},SSE);
echo on
clc
% 对线性神经网络进行训练
net.trainParam.goal=0.001; % 设置训练目标误差
[net,tr]= train(net,P,T);
pause;
clc
% 在误差曲面上绘制当前误差位置点
SSE=sse(T-sim(net,P));
close(gcf);
plotep(net.IW{1,1},net.b{1},SSE);
pause
clc
% 绘制误差变化曲线
close(gcf);
plotperf(tr,net.trainParam.goal);
pause
clc
% 对线性神经网络进行仿真
A=sim(net,P)
```



```
% 计算仿真误差
E=T-A
SSE=sse(E)
echo off
```

例 4.8　利用线性神经网络求取非线性问题的最佳线性拟合解。前面已经知道，线性神经网络只能描述输入、输出间的线性映射关系，当输入、输出之间是非线性关系时，利用线性神经网络只能得到输入、输出间的最佳线性拟合解，而不能实现完全的线性拟合。本例我们将采用线性单神经元对如下定义的输入和目标矢量进行拟合，即

输入矢量为　$p = [1 \quad 1.2 \quad 3 \quad -1.2]$

目标矢量为　$t = [0.4 \quad 1 \quad 3 \quad -0.5]$

解：由输入矢量和目标矢量可见，两者之间是非线性关系，即找不到一组权值 W 和阈值 b 使得对所有的输入和目标样本矢量均满足线性表达式 $t = W*p + b$。本例中，我们将采用 newlind 函数直接设计法和 train 函数设计法两种方案对线性神经网络进行设计。完整的 MATLAB 程序如下：

```
% Example 4.8
%
close all
clear
clf reset
figure(gcf);
echo on
clc
% 定义输入和目标矢量
% P 为输入矢量
P=[1, 1.2, 3,  1.2];
% T 为目标矢量
T=[0.4, 1, 3, -0.5];
pause
clc
% 设定线性神经元可能的权值和阈值范围
w_range=-2:0.2:2;
b_range=-2:0.2:2;
% 根据输入和目标矢量绘制线性神经元的误差曲面
ES=errsurf(P,T,w_range,b_range,'purelin');
plotes(w_range,b_range,ES);
pause
echo off
```



```
clc
disp('1.采用 NEWLIND 函数直接设计线性神经网络');
disp('2.采用 TRAIN 函数训练并设计线性神经网络');
choice=input('请选择设计方案(1,2):');
if(choice==1)
    echo on
    clc
    % 用 NEWLIND 函数直接设计线性神经网络
    net=newlind(P,T);
    pause
    clc
    % 对线性神经网络进行仿真
    A=sim(net,P)
    % 计算仿真误差
    E=T-A
    SSE=sse(E)
    pause
    clc
    % 在误差曲面上绘制当前误差位置点
    figure(gcf);
    plotep(net.IW{1,1},net.b{1},SSE);
    pause
    clc
    echo off
    clf reset
    figure(gcf);
    echo on
    clc
    % 绘制拟合曲线
    plot(P,T,'r*');
    hold on;
    plot(P,A);
    pause
    clc
else
    echo on
    clc
    % 用 TRAIN 函数训练并设计线性神经网络
```



```
% 计算最快的稳定学习速率值
maxlr=maxlinlr(P,'bias')
pause
clc
% 创建线性神经网络
net=newlin(minmax(P),1,[0],0.5*maxlr);
% 在误差曲面上选择权值和阈值对线性神经元进行初始化
plotes(w_range,b_range,ES);
subplot(1,2,2);
[net.IW{1,1},net.b{1}]=ginput(1);
echo off
SSE=sse(T-sim(net,P));
plotep(net.IW{1,1},net.b{1},SSE);
echo on
clc
% 对线性神经网络进行训练
net.trainParam.goal=0.01;% 设置训练目标误差
[net,tr]= train(net,P,T);
pause;
clc
% 对线性神经网络进行仿真
A=sim(net,P)
% 计算仿真误差
E=T-A
SSE=sse(E)
pause
clc
% 在误差曲面上绘制当前误差位置点
close(gcf);
plotep(net.IW{1,1},net.b{1},SSE);
pause
clc
% 绘制误差变化曲线
close(gcf);
lotperf(tr,net.trainParam.goal);
pause
clc
% 绘制拟合曲线
```



```
        close(gcf);
        plot(P,T,'r*');
        hold on;
        plot(P,A);
        pause
        clc
    end
    echo off
```

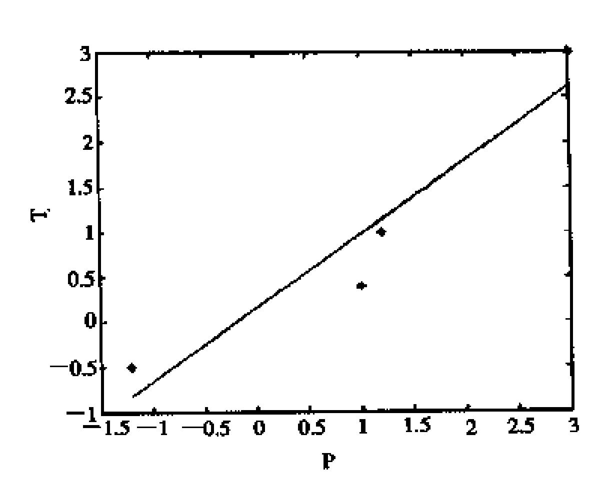


图 4.21 线性神经元最佳拟合曲线

实践证明，两种方案均未能达到拟合零误差的效果，但两种方案都得到了相同的最佳拟合结果，图 4.21 所示为拟合曲线。此外，图 4.22 给出了所设计的线性神经元的误差点位置。

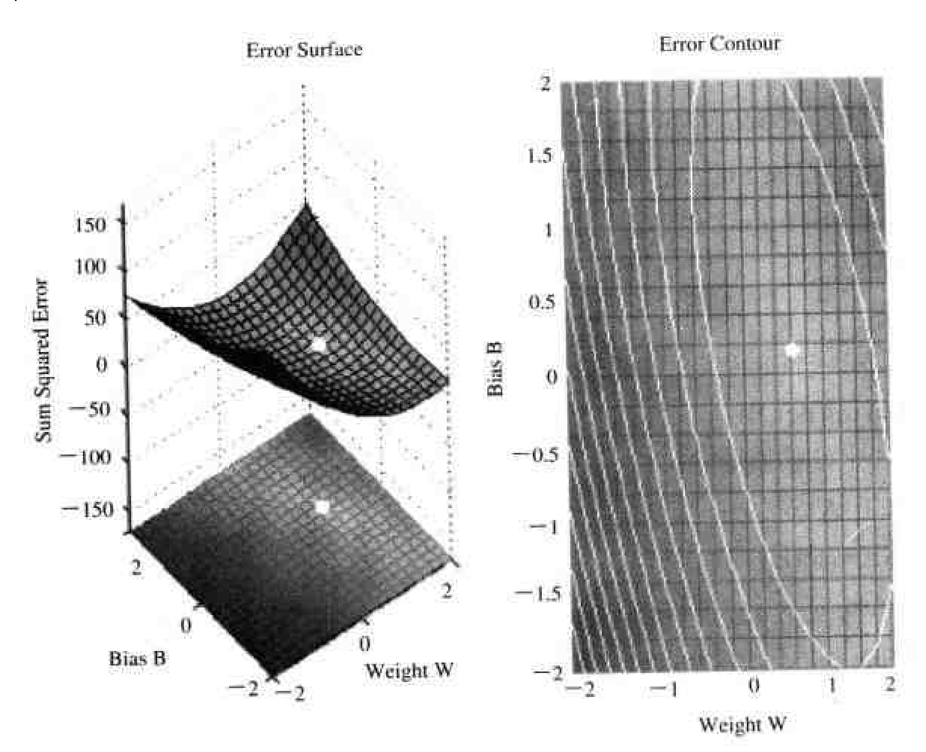


图 4.22 线性神经元的误差曲面和误差点

例 4.9 采用线性单神经元解决不确定性问题。我们已经知道，若线性神经网络的自由度大于训练样本集的个数，则对于样本匹配问题会得到无穷多个使训练误差为零的解，我们称此类问题为不确定性问题。本例中，采用线性单神经元对如下样本点进行匹配：

输入矢量为 p = [0.8]

目标矢量为 t = [−1.2]

解： 由所给样本点可见，对于线性单神经元，不存在数组权值 W 和阈值 b 使样本点满足线性表达式 t = W*p + b，所以该问题的解是不确定的。

本例的 MATLAB 程序与例 4.7 的几乎完全相同，只需要把输入矢量和目标矢量作相应修改即可。通过选取不同的网络初始权值和阈值，并利用 train 函数训练和设计线性神经



元，我们就可以得到不同的设计结果，但不论何种结果，线性神经元均能对训练样本完全匹配。图 4.23 和图 4.24 分别给出了两种不同设计的误差曲面和误差点结果。可见，尽管这两种设计结果的神经元所对应的误差点位于不同的位置，但都位于零误差带内。

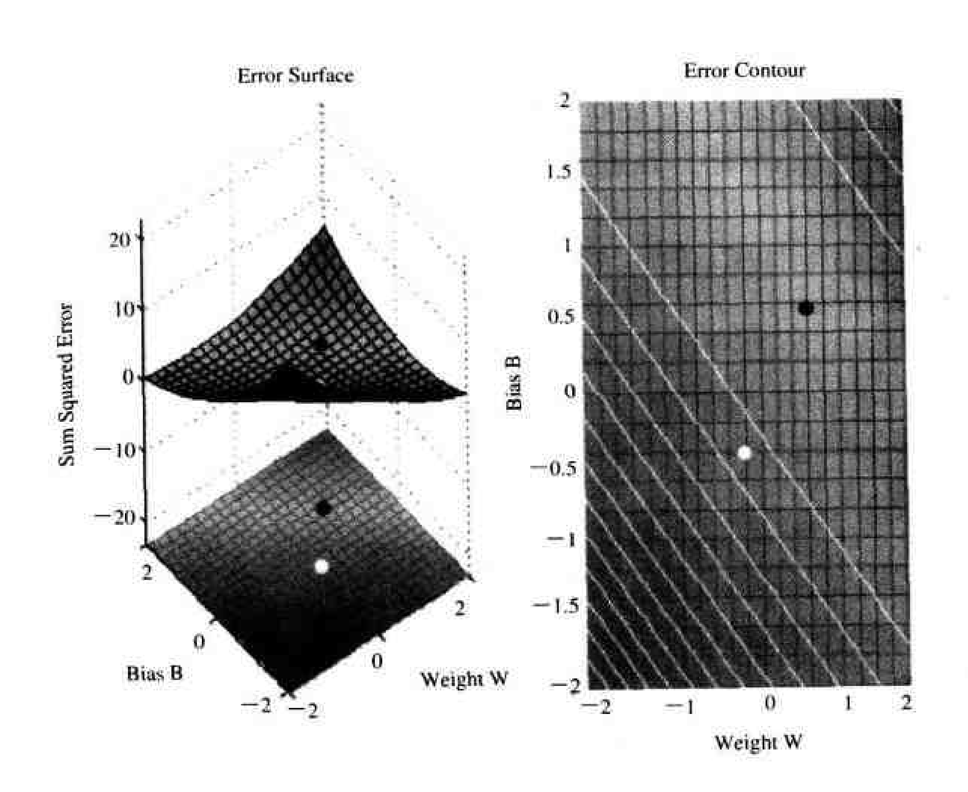


图 4.23 设计结果 1 的误差曲面和误差点

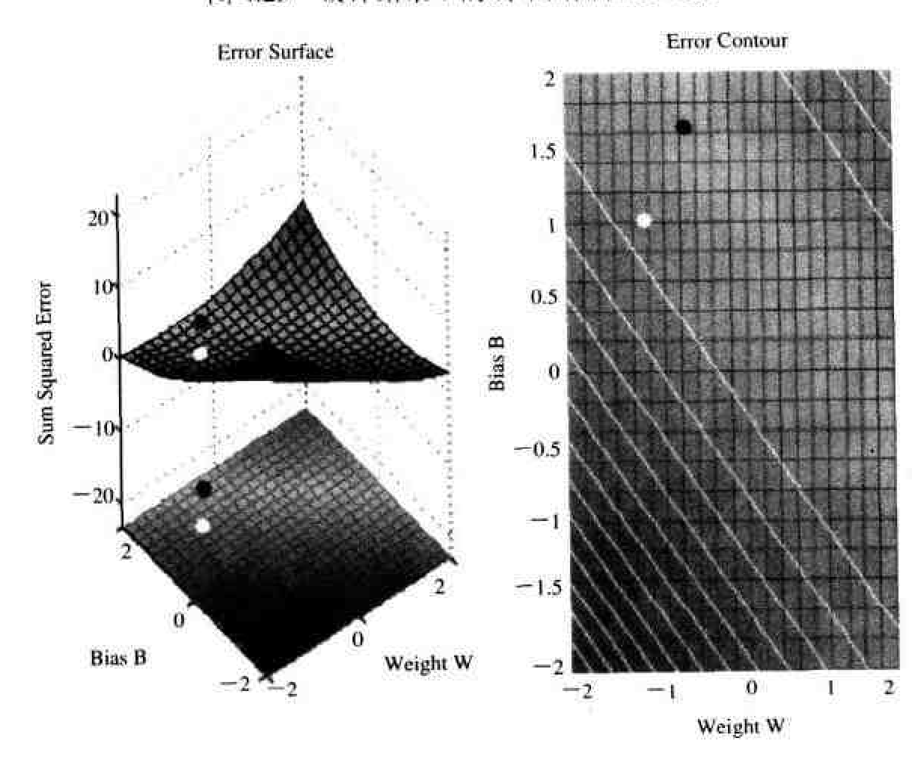


图 4.24 设计结果 2 的误差曲面和误差点

注：图中白点表示线性神经元的初始误差点，黑点表示训练后的误差点。



例 4.10 对于例 4.7 同样的问题，研究学习速率对线性神经网络训练结果的影响。

解：在使用 train 函数训练并设计线性单神经元时，如果采用比 maxlinlr 函数所计算的学习速率更大的值（如采用 2.2 倍于最大学习速率的值）对神经元进行训练，则会得到如图 4.25 所示的训练误差变化曲线。可见，若线性神经元学习速率过大，则会导致训练结果发散。

本例的 MATLAB 程序与例 4.7 的基本一样，只是要把下面的语句

net=newlin(minmax(P),1,[0],0.4*maxlr);

改为

net=newlin(minmax(P),1,[0],2.2*maxlr);

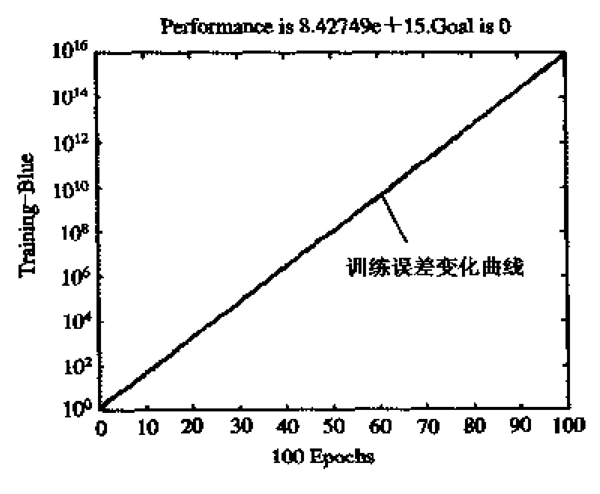


图 4.25 误差变化曲线

4.3 BP 神经网络的设计实例

例 4.11 采用动量梯度下降算法训练 BP 网络。训练样本定义如下：

输入矢量为 $p=\begin{bmatrix}-1 & -2 & 3 & 1\\ -1 & 1 & 5 & -3\end{bmatrix}$

目标矢量为 $t=[-1\ \ -1\ \ 1\ \ 1]$

解：本例的 MATLAB 程序如下：

```
% Example 4.11
%
close all
clear
echo on
clc
% NEWFF——生成一个新的前向神经网络
% TRAIN——对 BP 神经网络进行训练
% SIM——对 BP 神经网络进行仿真
pause      % 敲任意键开始
clc
% 定义训练样本
% P 为输入矢量
P=[-1, -2,  3,  1;
   -1,  1,  5, -3];
% T 为目标矢量
```

```
T=[-1, -1, 1, 1];
pause;
clc
% 创建一个新的前向神经网络
net=newff(minmax(P),[3,1],{'tansig','purelin'},'traingdm');
% 当前输入层权值和阈值
inputWeights=net.IW{1,1}
inputbias=net.b{1}
% 当前网络层权值和阈值
layerWeights=net.LW{2,1}
layerbias=net.b{2}
pause
clc
% 设置训练参数
net.trainParam.show = 50;
net.trainParam.lr = 0.05;
net.trainParam.mc = 0.9;
net.trainParam.epochs = 1000;
net.trainParam.goal = 1e-3;
pause
clc
% 调用 TRAINGDM 算法训练 BP 网络
[net,tr]=train(net,P,T);
pause
clc
% 对 BP 网络进行仿真
A = sim(net,P)
% 计算仿真误差
E = T - A
MSE=mse(E)
pause
clc
echo off
```

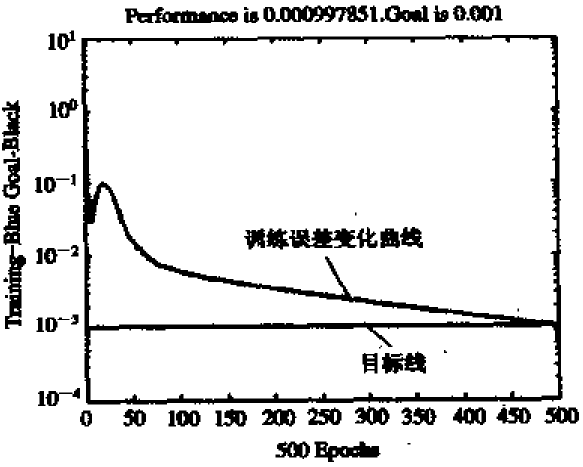


图 4.26 误差变化曲线

下面是本程序的某次训练结果。图 4.26 给出了相应的误差变化曲线，可见，当训练至第 500 步时，网络性能达标。

```
TRAINGDM, Epoch 0/1000, MSE 1.35673/0.001, Gradient 3.82647/1e-010
TRAINGDM, Epoch 50/1000, MSE 0.0158379/0.001, Gradient 0.112933/1e-010
TRAINGDM, Epoch 100/1000, MSE 0.00588767/0.001, Gradient 0.0289309/1e-010
TRAINGDM, Epoch 150/1000, MSE 0.00431032/0.001, Gradient 0.0208763/1e-010
TRAINGDM, Epoch 200/1000, MSE 0.00337422/0.001, Gradient 0.0173309/1e-010
```



TRAINGDM, Epoch 250/1000, MSE 0.00270622/0.001, Gradient 0.0149965/1e-010
TRAINGDM, Epoch 300/1000, MSE 0.00219738/0.001, Gradient 0.0132451/1e-010
TRAINGDM, Epoch 350/1000, MSE 0.00179668/0.001, Gradient 0.0118326/1e-010
TRAINGDM, Epoch 400/1000, MSE 0.00147491/0.001, Gradient 0.0106505/1e-010
TRAINGDM, Epoch 450/1000, MSE 0.00121304/0.001, Gradient 0.00963926/1e-010
TRAINGDM, Epoch 500/1000, MSE 0.00099785/0.001, Gradient 0.00875812/1e-010
TRAINGDM, Performance goal met.

例 4.12　采用 L-M 优化算法训练 BP 网络。训练样本同例 4.11。

解：本例的 MATLAB 程序与例 4.11 的基本相同，只是在生成前向网络时，要把训练函数由 traingdm 函数改为 trainlm 函数。下面只给出本例网络的某次训练结果和误差变化曲线（如图 4.27 所示）。

TRAINLM, Epoch 0/1000, MSE 1.56314/0.001, Gradient 3.85214/1e-010
TRAINLM, Epoch 3/1000, MSE 7.18461e-007/0.001, Gradient 0.00244234/1e-010
TRAINLM, Performance goal met.

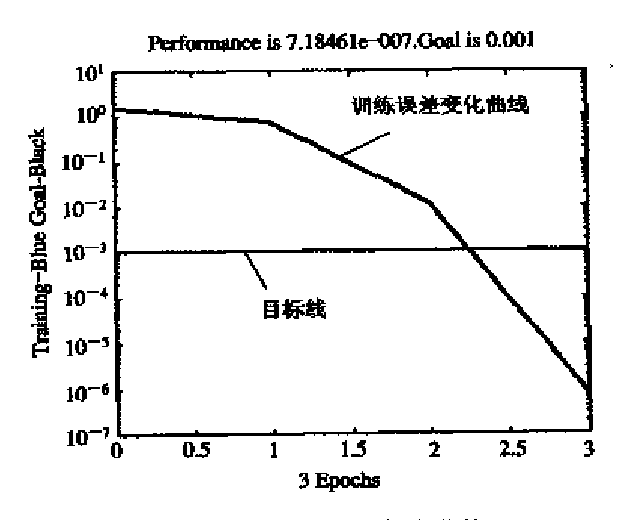


图 4.27　训练误差变化曲线

将本例训练结果与例 4.11 对比可见，L-M 优化算法的收敛速度明显优于动量梯度下降算法，而且 L-M 优化算法往往能得到更小的训练误差。

例 4.13　采用贝叶斯正则化算法提高 BP 网络的推广能力。在本例中，我们采用两种训练方法，即 L-M 优化算法（trainlm）和贝叶斯正则化算法（trainbr），用以训练 BP 网络，使其能够拟合某一附加有白噪声的正弦样本数据。其中，样本数据可以采用如下 MATLAB 语句生成：

输入矢量：P = [-1:0.05:1];

目标矢量：randn('seed',78341223);

　　T = sin(2*pi*P)+0.1*randn(size(P));

解：本例的 MATLAB 程序如下：

```
% Example 4.13
%
close all
```



```
clear
echo on
clc
% NEWFF——生成一个新的前向神经网络
% TRAIN——对 BP 神经网络进行训练
% SIM——对 BP 神经网络进行仿真
pause      % 敲任意键开始
clc
% 定义训练样本矢量
% P 为输入矢量
P = [-1:0.05:1];
% T 为目标矢量
randn('seed',78341223);
T = sin(2*pi*P)+0.1*randn(size(P));
% 绘制样本数据点
plot(P,T,'+');
echo off
hold on;
plot(P,sin(2*pi*P),':');      % 绘制不含噪声的正弦曲线
echo on
clc
pause
clc
% 创建一个新的前向神经网络
net=newff(minmax(P),[20,1],{'tansig','purelin'});
pause
clc
echo off
clc
disp('1. L-M 优化算法 TRAINLM');
disp('2. 贝叶斯正则化算法 TRAINBR');
choice=input('请选择训练算法(1,2):');
figure(gcf);
if(choice==1)
      echo on
      clc
      % 采用 L-M 优化算法 TRAINLM
      net.trainFcn='trainlm';
      pause
```



```
        clc
        % 设置训练参数
        net.trainParam.epochs = 500;
        net.trainParam.goal = 1e-6;
        net=init(net);      % 重新初始化
        pause
        clc
elseif(choice==2)
        echo on
        clc
        % 采用贝叶斯正则化算法 TRAINBR
        net.trainFcn='trainbr';
        pause
        clc
        % 设置训练参数
        net.trainParam.epochs = 500;
        randn('seed',192736547);
        net = init(net);      % 重新初始化
        pause
        clc
end
% 调用相应算法训练 BP 网络
[net,tr]=train(net,P,T);
pause
clc
% 对 BP 网络进行仿真
A = sim(net,P);
% 计算仿真误差
E = T - A;
MSE=mse(E)
pause
clc
% 绘制匹配结果曲线
close all;
plot(P,A,P,T,'+',P,sin(2*pi*P),':');
pause;
clc
echo off
```



通过采用两种不同的训练算法，我们可以得到如图 4.28 和图 4.29 所示的两种拟合结果。图中的实线表示拟合曲线，虚线代表不含白噪声的正弦曲线，“+”点为含有白噪声的正弦样本数据点。显然，经 trainlm 函数训练后的神经网络对样本数据点实现了“过度匹配”，而经 trainbr 函数训练的神经网络对噪声不敏感，具有较好的推广能力。

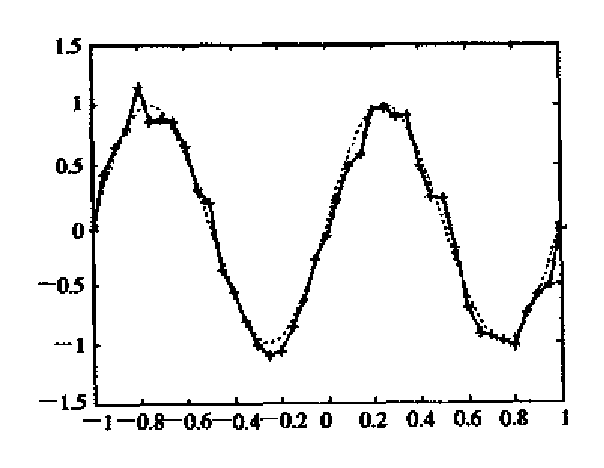


图 4.28 trainlm 训练后的拟合结果

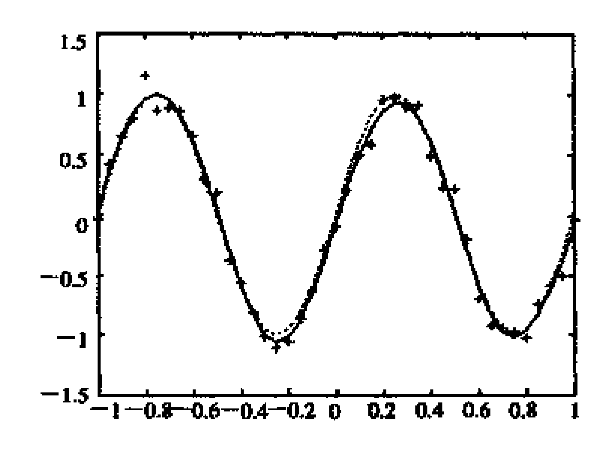


图 4.29 trainbr 训练后的拟合结果

值得指出的是，在利用 trainbr 函数训练 BP 网络时，若训练结果收敛，通常会给出提示信息“Maximum MU reached”。此外，用户还可以根据 SSE 和 SSW 的大小变化情况来判断训练是否收敛：当 SSE 和 SSW 的值在经过若干步迭代后处于恒值时，则通常说明网络训练收敛，此时可以停止训练。图 4.30 给出了本例中利用 trainbr 函数训练 BP 网络的误差变化曲线。可见，当训练迭代至 320 步时，网络训练收敛，此时 SSE 和 SSW 均为恒值，当前有效网络的参数（有效权值和阈值）个数为 11.7973。

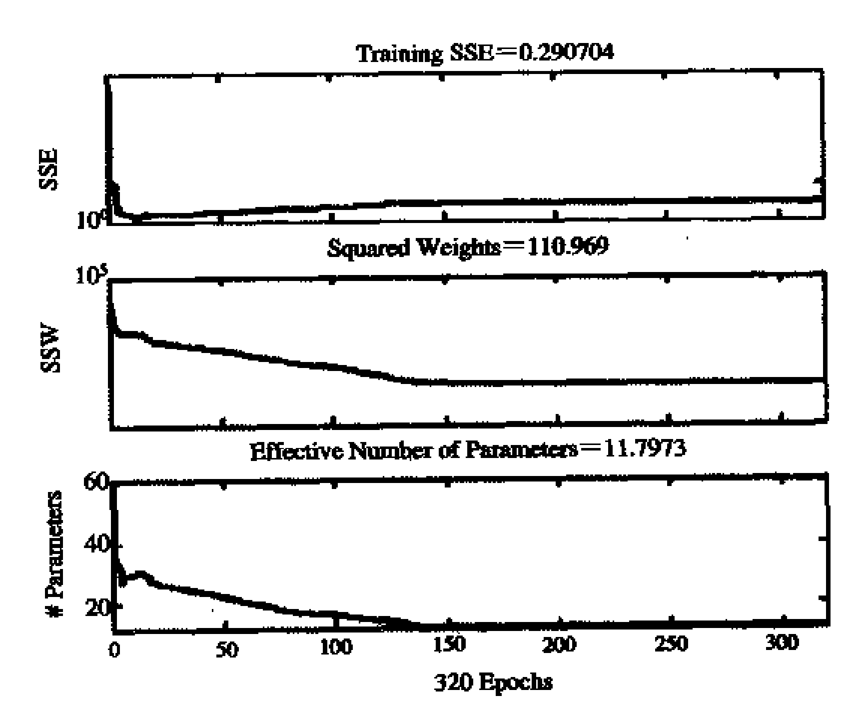


图 4.30 基于 trainbr 训练的误差变化曲线



例 4.14 采用“提前停止”方法提高 BP 网络的推广能力。对于和例 4.13 相同的问题，在本例中我们将采用训练函数 traingdx 和“提前停止”相结合的方法来训练 BP 网络，以提高 BP 网络的推广能力。

解：在利用“提前停止”方法时，首先应分别定义训练样本、验证样本或测试样本，其中，验证样本是必不可少的。在本例中，我们只定义并使用验证样本，即有

验证样本输入矢量：val.P = [-0.975:.05:0.975]

验证样本目标矢量：val.T = sin(2*pi*val.P)+0.1*randn(size(val.P))

值得注意的是，尽管“提前停止”方法可以和任何一种 BP 网络训练函数一起使用，但是不适合同训练速度过快的算法联合使用，比如 trainlm 函数，所以本例中我们采用训练速度相对较慢的变学习速率算法 traingdx 函数作为训练函数。

本例的 MATLAB 程序如下：

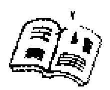
```
% Example 4.14
%
close all
clear
echo on
clc
% NEWFF——生成一个新的前向神经网络
% TRAIN——对 BP 神经网络进行训练
% SIM——对 BP 神经网络进行仿真
pause    % 敲任意键开始
clc
% 定义训练样本矢量
% P 为输入矢量
P = [-1:0.05:1];
% T 为目标矢量
randn('seed',78341223);
T = sin(2*pi*P)+0.1*randn(size(P));
% 绘制训练样本数据点
plot(P,T,'+');
echo off
hold on;
plot(P,sin(2*pi*P),':');      % 绘制不含噪声的正弦曲线
echo on
clc
pause
clc
% 定义验证样本
val.P = [-0.975:0.05:0.975];      % 验证样本的输入矢量
```



```
val.T = sin(2*pi*val.P)+0.1*randn(size(val.P));      % 验证样本的目标矢量
pause
clc
% 创建一个新的前向神经网络
net=newff(minmax(P),[5,1],{'tansig','purelin'},'traingdx');
pause
clc
% 设置训练参数
net.trainParam.epochs = 500;
net = init(net);
pause
clc
% 训练 BP 网络
[net,tr]=train(net,P,T,[],[],val);
pause
clc
% 对 BP 网络进行仿真
A = sim(net,P);
% 计算仿真误差
E = T - A;
MSE=mse(E)
pause
clc
% 绘制仿真拟合结果曲线
close all;
plot(P,A,P,T,'+',P,sin(2*pi*P),':');
pause;
clc
echo off
```

下面给出了网络的某次训练结果，可见，当训练至第 160 步时，训练提前停止，此时的网络误差为 0.0096035。图 4.31 给出了训练后的仿真数据拟合曲线，效果是相当满意的。

```
TRAINGDX, Epoch 0/500, MSE 1.88365/0, Gradient 3.06586/1e-006
TRAINGDX, Epoch 25/500, MSE 0.760054/0, Gradient 0.807722/1e-006
TRAINGDX, Epoch 50/500, MSE 0.389079/0, Gradient 0.420916/1e-006
TRAINGDX, Epoch 75/500, MSE 0.10162/0, Gradient 0.111724/1e-006
TRAINGDX, Epoch 100/500, MSE 0.0433248/0, Gradient 0.0341923/1e-006
TRAINGDX, Epoch 125/500, MSE 0.0255292/0, Gradient 0.0182261/1e-006
TRAINGDX, Epoch 150/500, MSE 0.00997371/0, Gradient 0.0380257/1e-006
```



```
TRAINGDX, Epoch 160/500, MSE 0.0096035/0, Gradient 0.0143/1e-006
TRAINGDX, Validation stop.
```

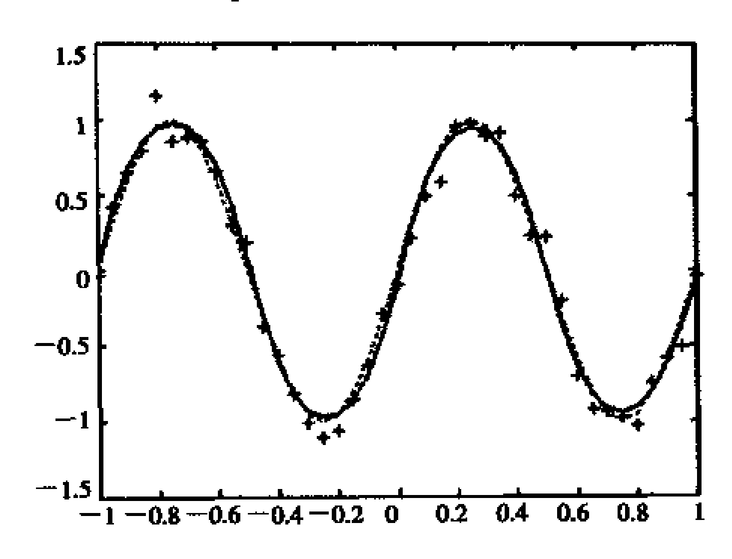


图 4.31 采用"提前停止"方法训练后的拟合曲线

4.4 径向基函数网络的设计实例

例 4.15 利用径向基函数网络实现函数逼近。

解：待逼近函数的表达式为

T = sin (5*P) + cos (3*P);

函数自变量的变化范围是

P = −1 : 0.1 : 1;

本例采用 newrb 函数设计径向基网络。下面给出MATLAB 程序：

```
% Example 4.15
%
close all
clf reset
figure (gcf);
echo on
clc
% NEWRB——设计径向基函数神经网络
% SIM——对径向基函数网络进行仿真
pause
clc
% P 是输入矢量，T 是目标矢量
P = -1 : 0.1 : 1;
T = sin (5*P) + cos (3*P);
```



```
% 画出待逼近函数的图形
plot (P, T, '+');
pause
clc
% 设计径向基函数网络，对函数进行逼近
goal = 0.01;             % 误差指标
sp = 1;                  % 扩展常数
mn = 20;                 % 神经元的最多个数
df = 1;                  % 训练过程的显示频率
net = newrb (P , T , goal , sp, mn , df );
pause
clc
% 对网络进行仿真，并画出样本数据和网络输出图形
Y = sim (net, P);
plot (P, T, '+');
hold on;
plot (P, Y);
echo off
```

程序运行结果如图 4.32 所示，图中的样本数据用符号"+"表示。

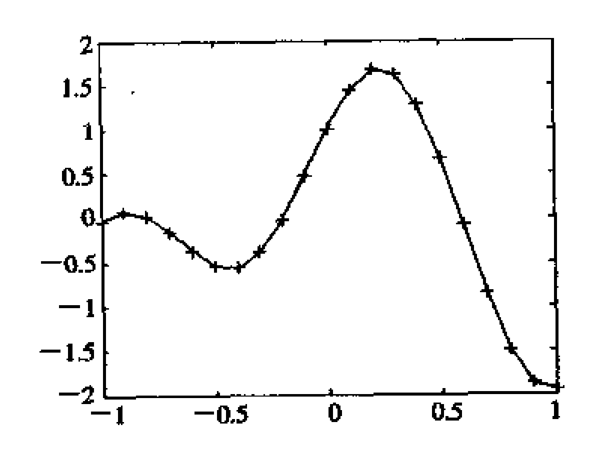


图 4.32　径向基网络的函数逼近结果

例 4.16　利用广义回归网络实现函数逼近。

解：待逼近函数的表达式为

```
T = cos (4*P) - sin (2*P) ;
```

函数自变量的变化范围是

```
P = -1 : 0.2 : 1;
```

利用newgrnn 函数设计广义回归网络：

```
net = newgrnn (P , T , sp);
```

在广义回归网络中，扩展常数 sp 的选取决定了网络的逼近性能，一般来说，扩展常



数应和输入数据的平均间距相当。本例中将选取两个扩展常数分别建立广义回归网络，并对网络的性能进行比较。下面给出设计广义回归网络的 MATLAB 程序：

```
% Example 4.16
%
close all
clf reset
figure (gcf);
echo on
clc
% NEWGRNN——设计广义回归网络
% SIM——对广义回归网络进行仿真
pause
clc
% P 是输入矢量，T 是目标矢量
P = -1 : 0.2 : 1;
T = cos (4*P) - sin (2*P) ;
% 画出待逼近函数的图形
plot (P, T, '+');
pause
clc
% 设计两个广义回归网络，对函数进行逼近
% 第一个网络
sp1 = 0.05;                        % 扩展常数 1
net1 = newgrnn (P , T ,  sp1);     % 网络 1
% 第二个网络
sp2 = 0.7;                         % 扩展常数 2
net2 = newgrnn (P , T ,  sp2);     % 网络 2
pause
clc
% 利用一组新数据对网络进行测试
P1 = -1.1 : 0.2 : 1.1;
% 对网络 1 进行仿真，并画出样本数据图形和网络输出图形
plot (P, T, '.', 'markersize', 20);
title ('net1');
hold on;
Y1 = sim (net1, P1);
plot (P1, Y1, '*', 'markersize', 10, 'color', [ 1 0 0 ]);
pause
```



```
clc
% 对网络 2 进行仿真，并画出样本数据图形和网络输出图形
figure;
plot (P, T, '.', 'markersize', 20);
hold on;
Y2 = sim (net2, P1);
title ('net2');
plot (P1, Y2, '*', 'markersize', 10, 'color', [ 1 0 0 ]);
echo off
```

程序运行结果如图 4.33 所示，图中的黑点标注的是样本数据，星号标注的是网络对测试数据的输出。从图中可以看到，由于网络 2 的扩展常数取得过大，因此该网络对数据的细节变化部分不能成功的逼近。

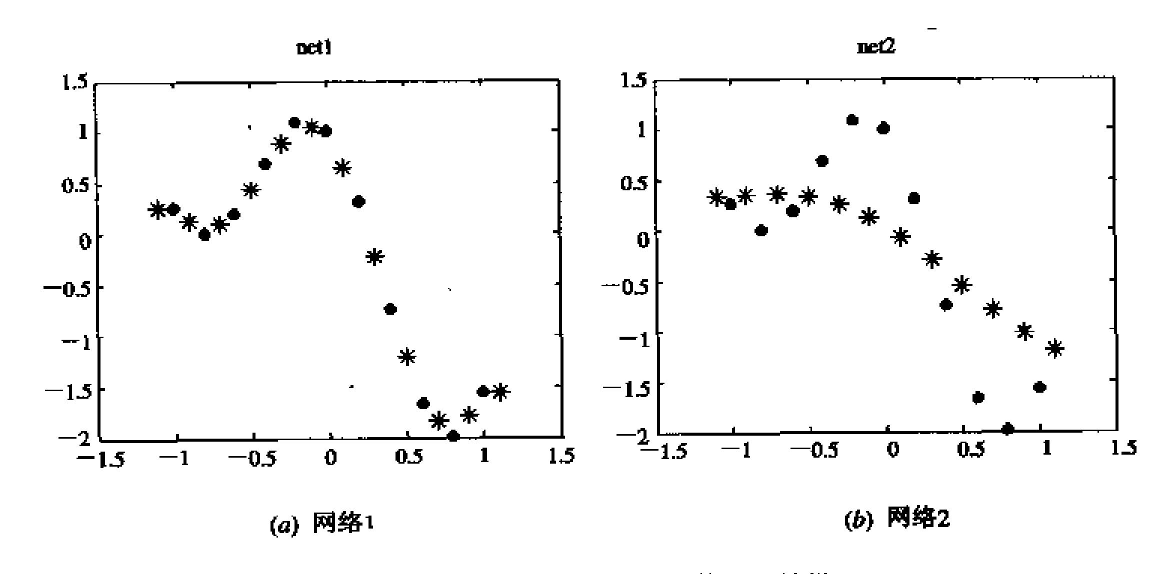


图 4.33　广义回归网络的函数逼近结果

例 4.17　利用概率神经网络对样本数据进行分类。

解：待分类的样本数据为

$$P=\begin{bmatrix}1 & -2 & -1 & 2 & 2 & -2\\ 2 & -1 & -1 & 2 & 1 & -2\end{bmatrix}$$

表示样本数据类别的下标矩阵为

$$T1=\begin{bmatrix}1 & 2 & 2 & 1 & 1 & 2\end{bmatrix}$$

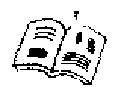
概率神经网络的输出形式是单值矢量组，因此首先应把下标矩阵转换为单值矢量组：

```
T = ind2vec (T1);
```

然后利用 newpnn 函数建立概率神经网络：

```
net = newpnn (P , T, sp);
```

本例完整的 MATLAB 程序如下：



```
% Example 4.17
%
close all
clf reset
figure (gcf);
echo on
clc
% NEWPNN——设计概率神经网络
% SIM——对概率神经网络进行仿真
pause
clc
% P 是样本数据，T 是表示样本数据类别的下标矩阵
P = [1   -2    -1    2    2    -2;
       2   -1    -1    2    1    -2];
T1 = [1    2     2     1    1     2];
% 将下标矩阵变为单值矢量组作为网络的目标输出
T = ind2vec (T1);
pause
clc
% 设计概率神经网络
sp = 1;       % 扩展常数
net = newpnn (P , T, sp);
pause
clc
% 对网络进行仿真，并画出分类结果
Y = sim (net, P);
Y1 = vec2ind (Y);
figure;
for i = 1: 6
      if   Y1(i) == 1
             plot (P(1, i), P(2, i), '*', 'markersize', 10);
      else
             plot (P(1, i), P(2, i), '+', 'markersize', 10);
      end
      hold on;
end
```

```
axis ([-3   3   -3   3 ]);
title ('class 1: *          class 2: +');
pause
clc
% 对一组新的输入数据进行分类
P1 = [-1; -2];
Y = sim (net, P1);
Y1 = vec2ind (Y)
echo off
```

分类结果如图 4.34 所示。图中星号标注的是原样本数据中的第一类数据，加号标注的是第二类数据。

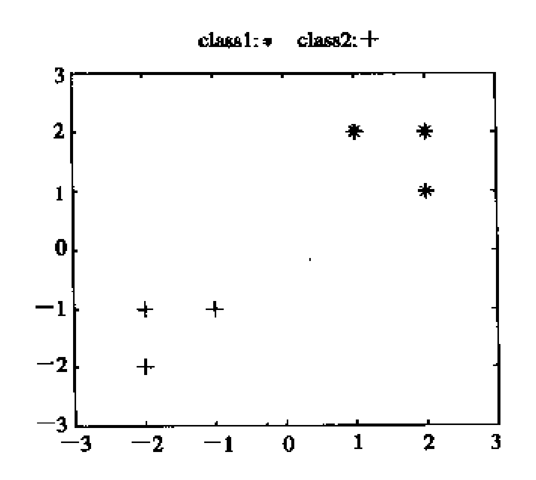


图 4.34　概率神经网络的分类结果

4.5　自组织网络的设计实例

例 4.18　利用竞争层网络对样本数据进行分类。

解：本例中待分类的样本数据由 nngenc 函数随机产生，即

```
P = nngenc (range, class, num, std);
```

其中，参数 class 表示样本数据的类别个数。然后利用 newc 函数建立竞争层网络：

```
net = newc (range , class , klr , clr);
```

其中，class 是数据类别个数，也是竞争层神经元的个数；klr 和 clr 分别是网络的权值学习速率和阈值学习速率。竞争层网络在训练时不需要目标输出，网络通过对数据分布特性的学习，自动地将数据划分为指定类别数。网络的训练语句如下（其中，默认的训练函数为 trainr）：

```
net = train (net, P);
```

在对训练好的网络进行仿真时，网络的输出为单值矢量组，为了观察方便，一般要将单值矢量组转化为下标矩阵的形式：

```
Y = sim (net, P);
Y1 = vec2ind (Y);
```

本例完整的 MATLAB 程序如下：

```
% Example 4.18
%
close all
clf reset
figure (gcf);
echo on
clc
```

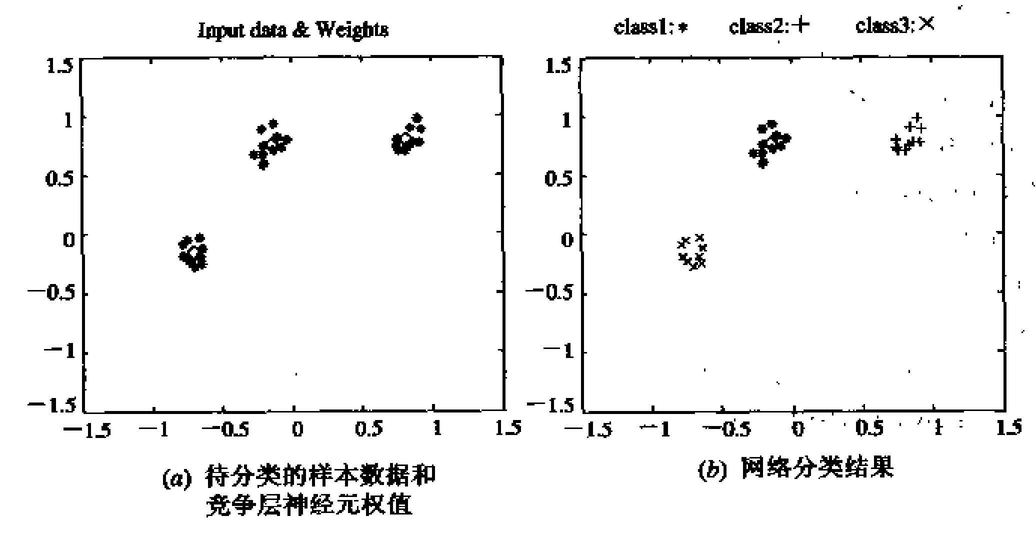


```
% NEWC——创建竞争层网络
% TRAIN——对竞争层网络进行训练
% SIM——对竞争层网络进行仿真
pause
clc
% 产生样本数据 P ，P 中包括三类共 30 个二维矢量
range = [-1 1; -1 1];                   % 样本数据取值范围
class = 3;                              % 样本数据类别数
num = 10;                               % 每类样本数据的个数
std = 0.1;                              % 每类样本数据的方差
P = nngenc (range, class, num, std);
pause
clc
% 画第一幅图：样本数据分布图
plot (P(1, :), P(2, :), '*', 'markersize', 5);    axis ([-1.5  1.5  -1.5  1.5]);
pause
clc
% 建立竞争层网络（由于样本数据分为三类，因此竞争层由三个神经元构成）
klr = 0.1;                              % 权值学习速率
clr =  0.01;                            % 阈值学习速率
net = newc (range , class , klr , clr);
pause
clc
% 对网络进行训练
net.trainParam.epochs = 5;              % 训练过程每五步显示一次
net = train (net, P);
pause
clc
% 在第一幅图上画出竞争层神经元权值，也就是每类样本数据的聚类中心
w = net.IW{1};
hold on;      plot (w(:, 1), w(:, 2), 'ob');
title ('Input data & Weights');
pause
clc
% 利用原始样本数据对网络进行仿真
Y = sim (net, P);
Y1 = vec2ind (Y)
pause
clc
% 画第二幅图：用不同符号标注数据分类结果
```




```
figure;
for i = 1: 30
    if   Y1(i) == 1
      plot (P(1, i), P(2, i), '*', 'markersize', 5);
    elseif   Y1(i) == 2
      plot (P(1, i), P(2, i), '+', 'markersize', 5);
    else
      plot (P(1, i), P(2, i), 'x', 'markersize', 5);
    end
    hold on;
end
axis ([-1.5   1.5   -1.5   1.5]);
title ('class 1: *     class 2: +     class 3: x');
pause
clc
% 利用一组新的输入数据检验网络性能
p = [-0.4   -0.4;   -0.1   0.9];
y = sim (net, p);
y1 = vec2ind (y)
echo off
```

程序运行结果如图 4.35 所示。在图 4.35(*a*)中，待分类的样本数据用星号标注，网络训练完毕后的竞争层神经元权值由圆圈标注。在图 4.35(*b*)中，已经划分好的三类数据分别用星号、加号和“x”符号标注。



(*a*) 待分类的样本数据和竞争层神经元权值　　(*b*) 网络分类结果

图 4.35　待分类的样本数据和竞争层网络的分类结果

例 4.19　利用一维自组织网络对样本数据进行分类。

解：本例中待分类的样本数据是 100 个分布在 1/4 圆弧上的矢量点。样本数据 P 采用如下方式产生：



```
angles = 0: 0.5*pi/99: 0.5*pi;
P = [ cos(angles); sin(angles) ];
```

图4.36(*a*)中绘出了样本数据的分布图。利用 newsom 函数建立自组织网络：

```
net = newsom ([0 1; 0 1], [9]);
```

该网络的竞争层共有 9 个神经元，即数据的类别个数。自组织网络在训练时不需要目标输出，网络通过对数据分布特性的学习，自动地将数据划分为指定的类别数。

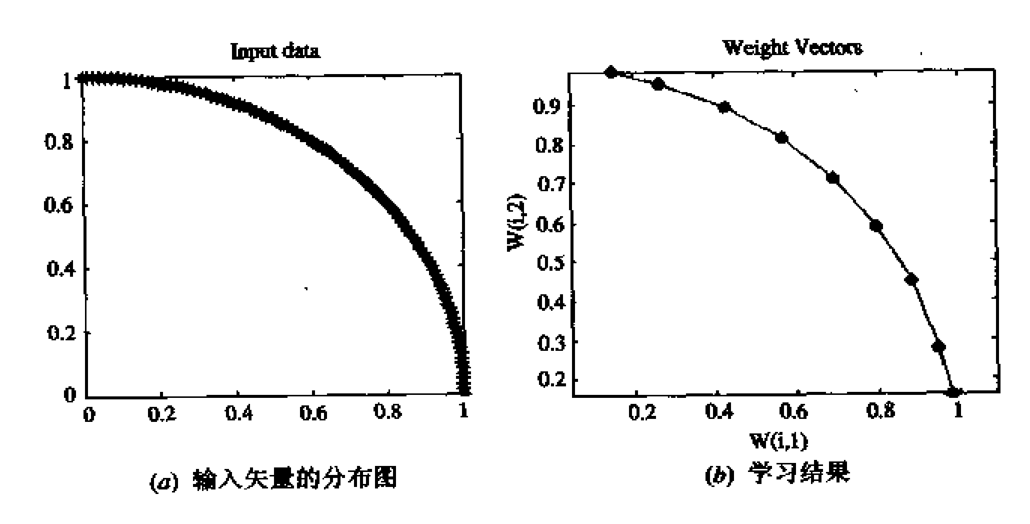
下面是完整的 MATLAB 程序：

```
% Example 4.19
%
close all
clf reset
figure (gcf);
echo on
clc
% NEWSOM——创建自组织网络
% TRAIN——对自组织网络进行训练
% SIM——对自组织网络进行仿真
pause
clc
% 产生样本数据 P
angles = 0: 0.5*pi/99: 0.5*pi;
P = [ cos(angles); sin(angles) ];
pause
clc
% 画第一幅图：样本数据分布图
plot (P(1, :), P(2, :), '*');
axis ([0   1   0   1]);
title ('Input data');
pause
clc
% 建立自组织网络
% 欲将样本数据分为 9 类，因此网络的竞争层由 9 个神经元构成
net = newsom ([0 1; 0 1], [9]);
pause
clc
% 对网络进行训练
net.trainParam.epochs = 10;
net = train (net, P);
pause
clc
```



```
% 画第二幅图：画出网络神经元权值，也就是每类样本数据的聚类中心
figure;
w = net.IW{1};
plotsom (net.IW{1, 1}, net.layers{1}.distances);
pause
clc
% 利用一组新的输入数据检验网络性能
a = sim (net, [0.6; 0.8])
echo off
```

程序运行结果如图4.36所示。从图4.36(*b*)中可以看到，自组织网络不仅能够实现数据的分类，还可以对输入数据的分布情况进行学习。



(*a*) 输入矢量的分布图　　(*b*) 学习结果

图4.36　一维自组织网络的输入矢量分布图及学习结果

例4.20　利用二维自组织网络对样本数据进行分类。

解：首先，随机产生待分类的样本数据：

```
P = rands (2, 500);
```

图4.37中绘出了输入样本矢量的分布图。然后利用 newsom 函数建立自组织网络：

```
net = newsom ([-1 1; -1 1], [4 5]);
```

该网络的竞争层共有 20 个神经元，即样本数据的类别个数，竞争层神经元的拓扑结构采用默认的六边形网格结构。下面给出完整的 MATLAB 程序：

```
% Example 4.20
%
close all
clf reset
figure (gcf);
echo on
clc
```

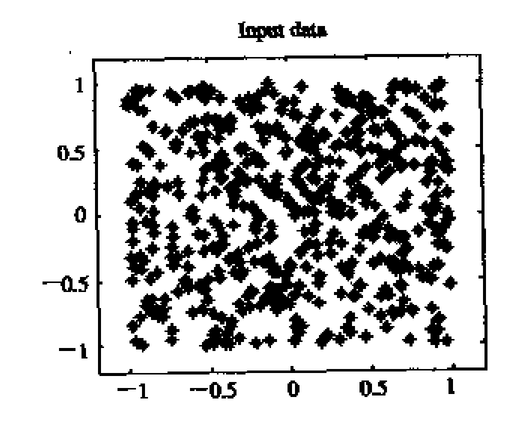


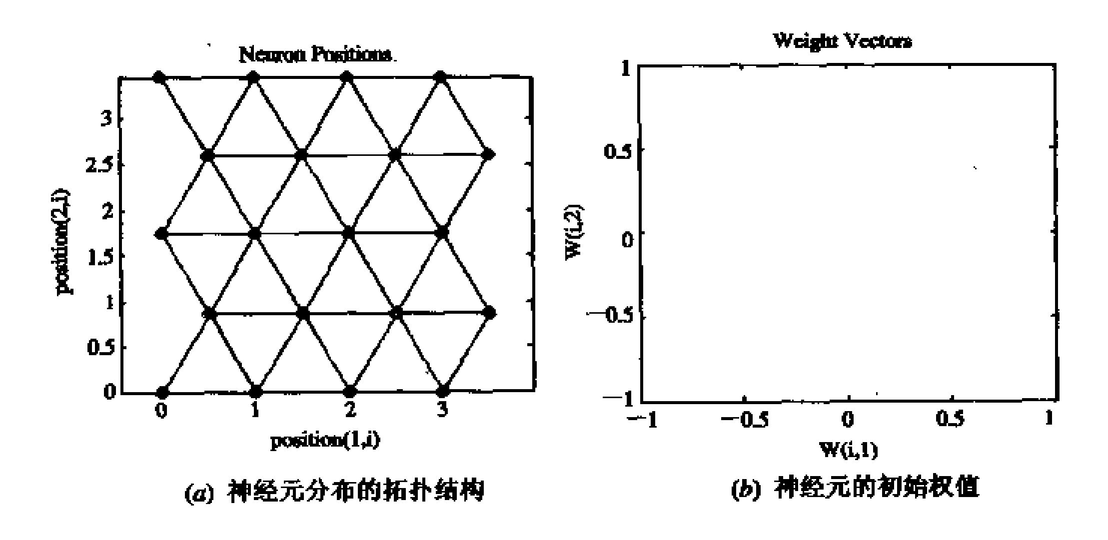
图4.37　二维自组织网络的输入矢量分布图



```
% NEWSOM——创建自组织网络
% TRAIN——对自组织网络进行训练
% SIM——对自组织网络进行仿真
pause
clc
% 随机产生样本数据 P
P = rands (2, 500);
pause
clc
% 画第一幅图：样本数据分布图
plot (P(1, :), P(2, :), '*');
axis ([-1.2   1.2   -1.2   1.2]);
title ('Input data');
pause
clc
% 建立自组织网络
% 欲将样本数据分为 20 类，因此网络的竞争层由 20 个二维分布的神经元构成
net = newsom ([-1 1; -1 1], [4 5]);
pause
clc
% 画第二幅图：神经元分布的拓扑结构图
figure;
plotsom (net.layers{1}.positions);
pause
clc
% 画第三幅图：网络初始状态下神经元权值的分布图
figure;
plotsom (net.IW{1, 1}, net.layers{1}.distances);
pause
clc
% 对网络进行训练
net.trainParam.epochs = 1;
net = train (net, P);
pause
clc
% 画第四幅图：画出网络神经元权值，也就是每类样本数据的聚类中心
figure;
plotsom (net.IW{1, 1}, net.layers{1}.distances);
pause
clc
% 画第五幅图：画出再次训练后的神经元权值
net.trainParam.epochs = 3;
```

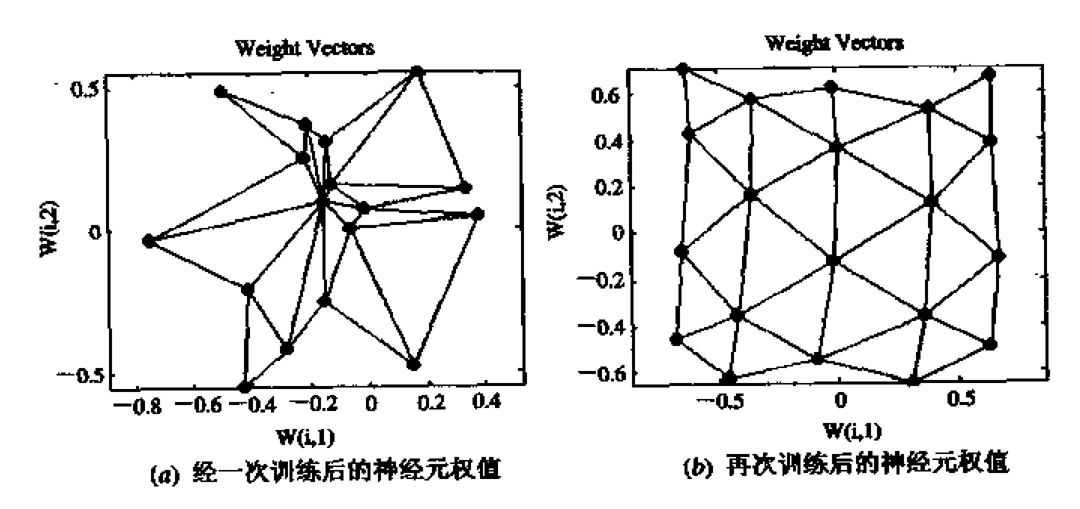
```
net = train (net, P);
figure;
plotsom (net.IW{1, 1}, net.layers{1}.distances);
pause
clc
% 利用一组新的输入数据检验网络性能
a = sim (net, [0.1; -0.5])
echo off
```

程序运行结果如图 4.37～图 4.39 所示。从图中可以看到自组织网络对输入数据分布情况的学习能力。



(*a*) 神经元分布的拓扑结构　(*b*) 神经元的初始权值

图 4.38　二维自组织网络的拓扑结构和神经元初始权值



(*a*) 经一次训练后的神经元权值　(*b*) 再次训练后的神经元权值

图 4.39　二维自组织网络的神经元权值在训练过程中的变化情况

例 4.21　利用学习矢量量化网络对样本数据进行分类。

解：待分类的样本数据为

$$P=\begin{bmatrix} -3 & -2 & -2 & 0 & 0 & 0 & 0 & 2 & 2 & 3 \\ 0 & 1 & -1 & 2 & 1 & -1 & -2 & 1 & -1 & 0 \end{bmatrix}$$





表示样本数据类别的下标矩阵为

$$Tc = [1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 2 \quad 2 \quad 1 \quad 1 \quad 1]$$

由于学习矢量量化网络的输出形式是单值矢量组，因此首先应把下标矩阵转换为单值矢量组的形式：

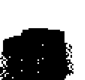
```
T = ind2vec (Tc);
```

然后利用 newlvq 函数建立学习矢量量化网络，网络的竞争层有 4 个神经元。两类数据在样本数据中所占的比例分别为0.6和0.4，学习速率设为0.1。

```
net = newlvq (minmax(P), 4, [0.6  0.4], 0.1);
```

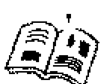
本例完整的 MATLAB 程序如下：

```
% Example 4.21
%
close all
clf reset
figure (gcf);
echo on
clc
% NEWLVQ——创建学习矢量量化网络
% TRAIN——对学习矢量量化网络进行训练
% SIM——对学习矢量量化网络进行仿真
pause
clc
% P 是样本数据，T 是表示样本数据类别的下标矩阵
P = [ -3 -2 -2 0 0 0 0 2 2 3;
      0 1 -1 2 1 -1 -2 1 -1 0];
Tc = [1 1 1 2 2 2 2 1 1 1];
% 将下标矩阵变为单值矢量组作为网络的目标输出
T = ind2vec (Tc);
pause
clc
% 建立学习矢量量化网络
% 网络竞争层有 4 个神经元，两类数据所占的比例分别为 0.6 和 0.4 ，学习速
  率为 0.1
net = newlvq (minmax(P), 4, [0.6  0.4], 0.1);
pause
clc
% 对网络进行训练
```






```
net.trainParam.epochs = 20;
net = train (net, P, T);
pause
clc
% 画出样本数据分布图
figure;
for i = 1: 10
     if   Tc(i) == 1
          plot (P(1, i), P(2, i), '*', 'markersize', 10);
     else
          plot (P(1, i), P(2, i), '+', 'markersize', 10);
     end
     hold on;
end
pause
clc
% 画出竞争层神经元的权值
w1 = net.IW{1}';
wc = vec2ind (net.LW{2});
for i = 1: 4
     if   wc(i) == 1
          plot (w1(1, i), w1(2, i), 'o', 'markersize', 5);
     else
          plot (w1(1, i), w1(2, i), 's', 'markersize', 5);
     end
     hold on;
end
axis ([-4   4   -3   3]);
title ('Input data & Net Weights');
pause
clc
% 利用样本数据对网络进行仿真
Y = sim (net, P);
Yc = vec2ind (Y);
pause
clc
% 利用一组新数据对网络进行检验
```



```
pcheck = [0; 0.5];
Y1 = sim (net, pcheck);
Yc1 = vec2ind (Y1)
echo off
```

程序运行结果如图 4.40 所示。图中星号表示第一类样本数据，加号表示第二类样本数据，圆圈和方块表示四个神经元的权值。其中，圆圈表示的神经元将对第一类样本数据产生高输出，方块表示的神经元将对第二类样本数据产生高输出。

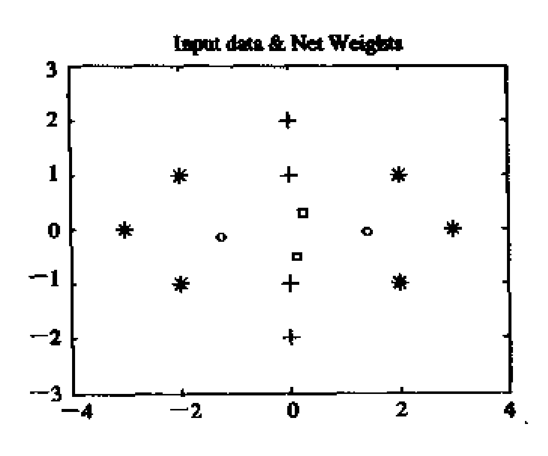


图 4.40　学习矢量量化网络的学习结果

4.6　Hopfield 网络的设计实例

例 4.22　设计一个含有两个神经元的 Hopfield 网络。

解：在此仅给出完整的 MATLAB 程序：

```
% Example 4.22
%
close all
clf reset
figure (gcf);
echo on
clc
% NEWHOP——创建 Hopfield 网络
% SIM——对 Hopfield 网络进行仿真
pause
clc
% 定义网络的两个稳定点
T = [1  -1;  -1  1]';
% 画出 hopfield 网络的状态空间和两个稳定点
plot (T(1, :), T(2, :),  '*');
axis ([-1.1  1.1  -1.1  1.1]);
pause
clc
% 建立 Hopfield 网络
net = newhop (T);
pause
clc
```

```
% 随机产生 20 个输入，对网络进行仿真
for   i = 1: 20
    a = {rands(2, 1)};
    [y, Pf, Af] = sim (net, {1 20}, {}, a);
    record = [cell2mat(a) cell2mat(y)];
    start = cell2mat (a);
    hold on;
    % 画出网络状态的变化轨迹
    plot (start(1,1), start(2,1), 'x', record(1, :),
    record(2, :));
end
echo off
```

图 4.41　两神经元 Hopfield 网络的联想记忆结果

程序运行结果如图 4.41 所示，图中的“x”符号表示网络的各初始状态。

例 4.23　设计一个含有三个神经元的 Hopfield 网络。

解：在此仅给出完整的 MATLAB 程序：

```
% Example 4.23
%
close all
clf reset
figure (gcf);
echo on
clc
% NEWHOP——创建 Hopfield 网络
% SIM——对 Hopfield 网络进行仿真
pause
clc
% 定义网络的两个稳定点
T = [1   -1   -1; 1   1   -1]';
pause
clc
% 画出 hopfield 网络的状态空间和两个稳定点
plot3 (T(1, :), T(2, :), T(3, :), '*');
axis ([-1.1   1.1   -1.1   1.1   -1.1   1.1]);      axis manual;
set (gca, 'box', 'on');
view ([37.5   30]);
pause
clc
% 建立 Hopfield 网络
net = newhop (T);
```



```
pause
clc
% 随机产生 20 个输入，对网络进行仿真
for  i = 1:20
   a = {rands(3, 1)};
   [y, Pf, Af] = sim (net, {1 20}, {}, a);
   record = [cell2mat(a) cell2mat(y)];
   start = cell2mat (a);
   hold on;
   % 画出网络状态的变化轨迹
   plot3 (start(1,1), start(2,1), start(3,1),
   'x', record(1, :), record(2, :), record(3, :));
end
echo off
```

程序运行结果如图 4.42 所示，图中的“x”符号表示网络的各初始状态。

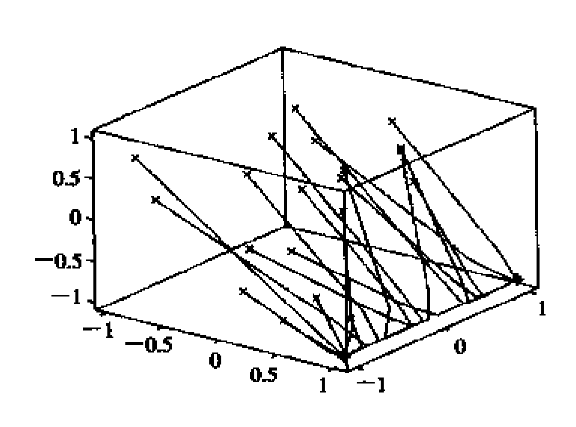


图 4.42 三个神经元 Hopfield 网络的联想记忆结果

4.7 神经网络的应用实例

神经网络以其强大的非线性映射能力和学习能力，能够解决许多在实际应用中采用常规方法难以处理的问题，因此神经网络在很多领域都得到了广泛的应用。目前，神经网络主要用于解决下列几类问题：

- 智能信息处理和模式识别。
- 函数拟合与系统辨识。
- 信号预测和信号处理。
- 预测与控制。

本节将介绍神经网络在预测、识别、控制和信号检测等方面的几个应用实例，从而使读者能够对神经网络在各领域的应用有个大致的了解，同时也希望为某些领域的实际系统设计与仿真提供参考。

4.7.1 线性神经网络在线性预测中的应用

例 4.24 利用线性神经网络对某一正弦信号进行线性预测。利用函数 newlind 设计线性神经网络，在已知正弦信号过去 5 个值的情况下，预测其将来值。

(1) 问题描述。

首先，待预测的正弦信号 T 可以通过如下的 MATLAB 语句生成：

```
time = 0:0.025:5;
T = sin (4*pi*time);
```



可见，所定义的正弦信号特征为：持续时间 5 秒，采样频率为 40 次 / 秒。调用 plot 函数，我们可以绘制出所定义的正弦信号曲线，如图 4.43 所示。

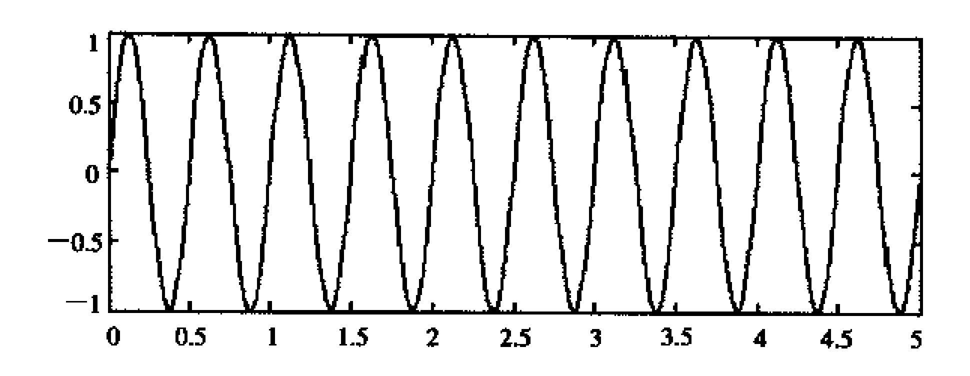


图 4.43　待预测的正弦信号曲线

由于本例的预测问题是采用正弦信号的前 5 个信号值来预测下一个信号值的，所以神经网络输入矢量 P 的 5 个分量可以通过将信号 T 分别延迟 1～5 s 来得到，即

```
Q = length(T);
P = zeros(5,Q);
P(1,2:Q) = T(1:(Q-1));
P(2,3:Q) = T(1:(Q-2));
P(3,4:Q) = T(1:(Q-3));
P(4,5:Q) = T(1:(Q-4));
P(5,6:Q) = T(1:(Q-5));
```

(2) 网络设计。

在定义了网络输入和目标信号矢量以后，就可以直接利用 newlind 函数对线性网络进行设计了。利用 newlind 函数可以求得最优的线性网络权值和阈值，即

```
net = newlind(P,T);
```

所生成的线性神经网络结构如图 4.44 所示。

(3) 网络测试。

利用 sim 函数可以对设计好的神经网络进行仿真，然后将仿真结果（预测输出）与实际信号进行比较，得出预测误差，即

```
a = sim (net , P);
e = T - a;
```

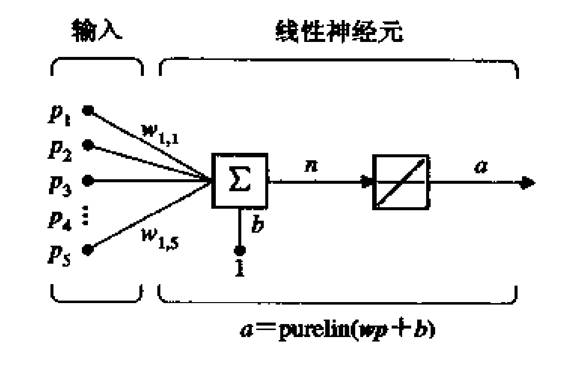
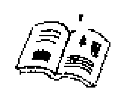


图 4.44　线性神经网络结构

图 4.45 和图 4.46 分别给出了预测输出曲线与实际信号曲线的比较图和预测误差曲线。由图 4.45 不难看出，信号的预测值和实际值相当接近；由图 4.46 可以看出，在初始阶段，误差较大，但经过一小段时间后，误差几乎趋于零。这是由于在起始的 5 个仿真步长内，网络需要的 5 个延迟输入并不完整，因此，在开始时出现初始误差是在所难免的。



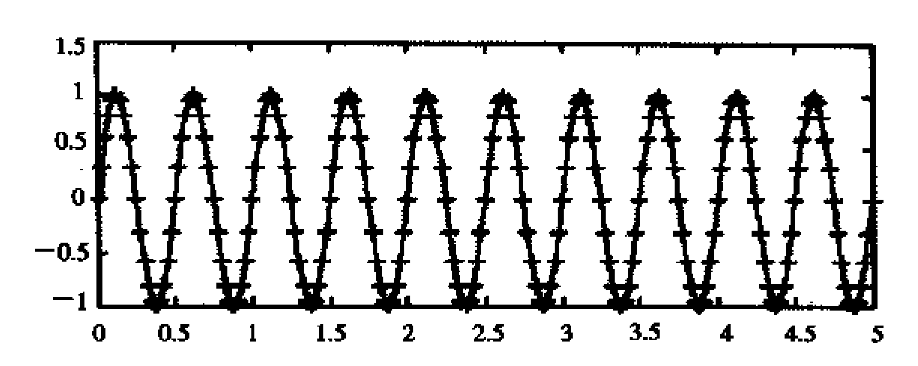


图 4.45　正弦信号曲线与网络预测输出曲线

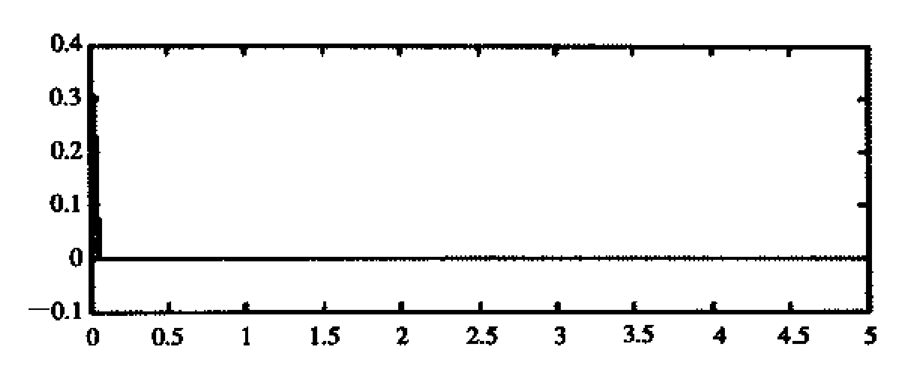


图 4.46　网络预测误差曲线

(4) 给出本例的 MATLAB 程序：

```
% Example 4.24
%
close all
clear

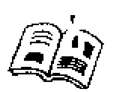
clf reset
figure(gcf);
echo on
clc
%利用线性神经网络对正弦信号进行线性预测
% NEWLIND——采用直接法设计线性神经网络
% SIM——对线性神经网络进行仿真
pause     % 敲任意键开始
clc
% 定义待预测的正弦信号
time = 0:0.025:5;
T = sin (4*pi*time);
% 绘制正弦信号曲线
plot(time,T);
```



```
pause
clc
% 根据待预测信号生成输入信号矢量
Q = length(T);
P = zeros(5,Q);
P(1,2:Q) = T(1:(Q-1));
P(2,3:Q) = T(1:(Q-2));
P(3,4:Q) = T(1:(Q-3));
P(4,5:Q) = T(1:(Q-4));
P(5,6:Q) = T(1:(Q-5));
pause
clc
% 设计线性神经网络
net=newlind(P,T);
pause
clc
% 对网络性能进行测试
a = sim (net,P);
e = T-a;
% 绘制网络预测输出曲线
hold on
plot(time,a,'r');
pause
clc
% 绘制预测误差曲线
echo off
clf reset
figure(gcf);
echo on
plot(time,e);
echo off
```

(5) 设计小结。

由本例可见，对于非线性信号的预测问题，利用线性神经网络虽然得不到使预测误差完全为零的解，但总可以找到一个使线性预测误差最小的解。而且实践证明，随着网络输入样本量的增加，线性网络的预测性能将大为提高。当然，如果涉及非线性较强的预测问题，并且所要求的预测误差指标又很高，那么利用线性神经网络的线性预测能力可能无法满足设计要求，这时采用BP网络或径向基函数网络将比较适合。



4.7.2 线性自适应滤波网络在自适应预测中的应用

例 4.25 利用 adapt 函数对自适应滤波网络进行训练，并用设计好的自适应滤波网络对某时变正弦信号进行自适应预测。

(1) 定义输入矢量和目标矢量。

待预测的时变正弦信号 T 定义如下：

$$T=\begin{cases}\sin(4\pi\ k) & k\le 4s\\ \sin(8\pi\ k) & 4s<k\le 6s\end{cases}$$

可见，4 秒以后正弦信号频率加倍。此外，信号的采样频率前 4 秒取每秒采样 20 次，4 秒以后加倍。该时变正弦信号 T 可以采用如下 MATLAB 语句生成：

```
time1 = 0:0.05:4;
time2 = 4.05:0.024:6;
time = [time1 time2];
T = [sin(time1*4*pi) sin(time2*8*pi)];
```

为了便于训练，将信号 T 转换成序列的形式：

```
T = con2seq(T);
```

令网络输入信号与目标序列相同

```
P = T;
```

(2) 网络设计。

同例 4.23，在每一个采样时刻点，我们采用该时刻之前的 5 个采样信号值作为网络的输入，网络的输出则为下一时刻信号的预测值。据此，线性自适应滤波网络可以按如图 4.47 所示的结构进行设计。

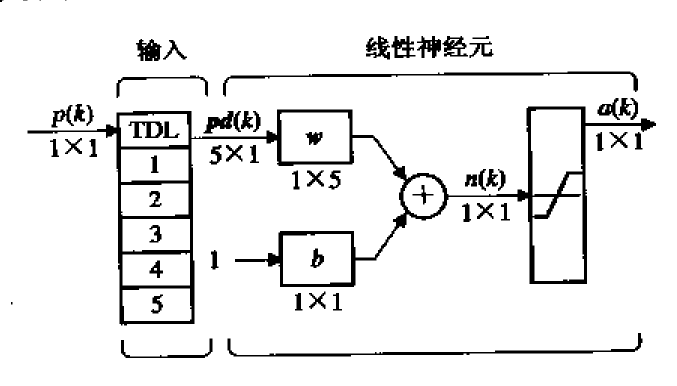


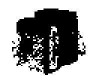
图 4.47 自适应滤波网络结构图

首先，采用 newlin 函数生成如图 4.47 所示的自适应滤波网络，即

```
lr = 0.1;
delays = [1 2 3 4 5];
net = newlin(minmax(cat(2,P{:})),1,delays,lr);
```

其中，网络学习速率 lr 取为 0.1。

然后，利用 adapt 函数对所生成的神经网络进行训练：



[net,a,e]=adapt(net,P,T);

训练完成之后，即返回设计好的自适应滤波网络。

(3) 网络测试。

为了检测网络的预测效果，在网络训练完成之后，可以将网络预测输出与目标信号绘制在同一幅图中加以比较，如图 4.48 所示，其中虚线代表目标信号。

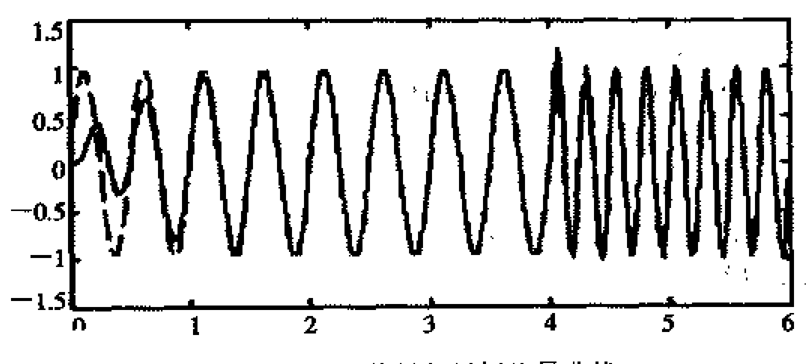


图 4.48　预测信号与目标信号曲线

由图 4.48 可见，在经过大约 1.5 秒的自适应训练之后，网络几乎可以完全跟踪目标信号（曲线几乎完全重合）。在第 4 秒时刻，由于目标信号的频率发生突变，所以网络的输出与目标信号曲线稍有偏差，但由于神经网络先前已经对与当前信号相似的正弦信号进行了学习，所以经过更短的训练时间就能再次精确地预测目标信号。

图 4.49 给出了网络的预测误差曲线。

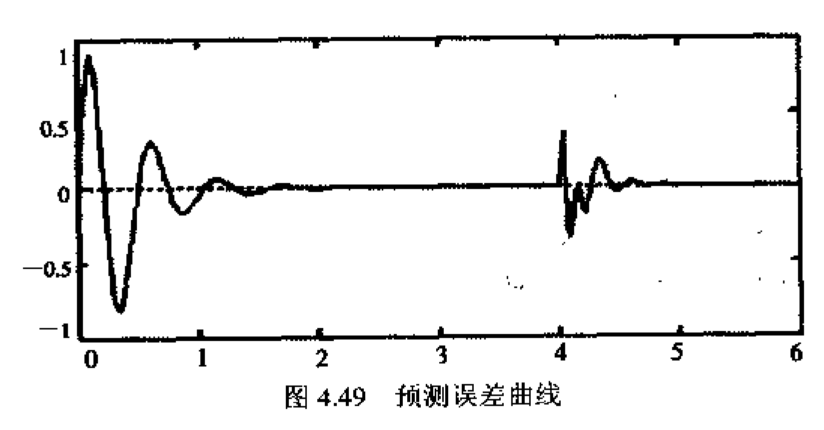


图 4.49　预测误差曲线

(4) 给出本例的 MATLAB 程序：

```
% Example 4.25
%
close all
clear
clf reset
figure(gcf);
echo on
clc
% 利用线性自适应滤波网络对时变信号进行自适应预测
% NEWLIN——生成一个新的线性自适应滤波网络
```



```
% ADAPT——对线性自适应滤波网络进行训练
pause    % 敲任意键开始
clc
% 定义输入矢量和目标矢量
time1 = 0:0.05:4;
time2 = 4.05:0.024:6;
time = [time1 time2];
T = [sin(time1*4*pi) sin(time2*8*pi)];
T = con2seq(T);     % 输入矢量
P = T;     % 目标信号
% 绘制待预测的目标信号曲线
plot(time,cat(2,T{:}))
pause
clc
% 生成自适应滤波网络
lr = 0.1;     % 学习速率
delays = [1 2 3 4 5];
net = newlin(minmax(cat(2,P{:})),1,delays,lr);
pause
clc
% 对网络进行训练
[net,y,e]=adapt(net,P,T);
pause
clc
% 绘制网络预测输出曲线
plot(time,cat(2,T{:}),'--',time,cat(2,y{:}),'r');
pause;
clc
% 绘制预测误差曲线
plot(time,cat(2,e{:}),[min(time) max(time)],[0 0],':r');
pause;
clc
echo off
```

4.7.3 BP 神经网络在模式识别中的应用

神经网络尤其是 BP 网络，以其强大的非线性映射能力，在模式识别领域得到了广泛的应用。本节我们将以 MATLAB 系统本身提供的两个典型实例为代表，具体讨论 BP 神经网络在模式识别领域的应用。

例 4.26 血清胆固醇含量检测问题。通常，医生总是根据某一血样的光谱成份来判定



血样中的血清胆固醇含量的高低（通常划分为 hdl、ldl、vldl 三个指标值）。通过临床实践共提取了 264 位病人的血样检测结果数据，其中每一个检测结果均对应所测血样光谱的 21 个波长值。现在要利用上述样本数据训练和设计一个神经网络，使其能够自动地完成上述检测过程。

(1) 样本数据的定义与预处理。

choles_all.mat 文件中存储了网络训练所需要的全部样本数据。利用 load 函数可以在工作空间中自动载入网络训练所需的输入数据 p 和目标数据 t，即

```
load choles_all
sizeofp = size (p)
sizeofp =
     21    264
sizeoft = size (t)
sizeoft =
     3    264
```

可见，样本集的大小为 264。为了提高神经网络的训练效率，通常要对样本数据作适当的预处理。首先，利用 prestd 函数对样本数据作归一化处理，使得归一化后的输入和目标数据均服从正态分布，即

```
[pn,meanp,stdp,tn,meant,stdt] = prestd(p,t);
```

然后，利用 prepca 函数对归一化后的样本数据进行主元分析，从而消除样本数据中的冗余成份，起到数据降维的目的。

```
[ptrans,transMat] = prepca(pn,0.001);
[R,Q] = size(ptrans)
R =
     4
Q =
     264
```

可见，主元分析之后的样本数据维数被大大降低，输入数据的维数由 21 变为 4。

(2) 对训练样本、验证样本和测试样本进行划分。

为了提高网络的推广能力和识别能力，训练中采用“提前停止”的方法，因此，在训练之前，需要将上面处理后的样本数据适当划分为训练样本集、验证样本集和测试样本集。样本数据集的划分可以通过如下 MATLAB 语句实现：

```
iitst = 2:4:Q;
iival = 4:4:Q;
iitr = [1:4:Q 3:4:Q];
val.P = ptrans(:,iival); val.T = tn(:,iival);       % 验证样本集
test.P = ptrans(:,iitst); test.T = tn(:,iitst);     % 测试样本集
ptr = ptrans(:,iitr); ttr = tn(:,iitr);             % 训练样本集
```

可见，验证样本和测试样本均是从原样本数据中均匀选取 1/4 而生成的。



(3) 网络生成与训练。

选用两层 BP 网络，其中网络输入维数为 4，输出维数为 3，输出值即为血清胆固醇的三个指标值大小。网络中间层神经元数目预选为 5，传递函数类型选为 tansig 函数，输出层传递函数选为线性函数 purelin，训练函数设为 trainlm。网络的生成语句如下：

```
net = newff(minmax(ptr),[5 3],{'tansig' 'purelin'},'trainlm');
```

利用 train 函数对所生成的神经网络进行训练，训练结果如下：

```
[net,tr]=train(net,ptr,ttr,[],[],val,test);
TRAINLM, Epoch 0/100, MSE 2.4689/0, Gradient 784.771/1e-010
TRAINLM, Epoch 20/100, MSE 0.328009/0, Gradient 4.33429/1e-010
TRAINLM, Validation stop.
```

可见，网络训练迭代至第 20 步时提前停止，这是由于验证误差已经开始变大。利用下面语句可以绘制出训练误差、验证误差和测试误差的变化曲线，如图 4.50 所示。由图可见，验证误差和测试误差的变化趋势基本一致，说明样本集的划分基本合理。由训练误差曲线可见，训练误差结果也是比较满意的。

```
plot(tr.epoch,tr.perf,tr.epoch,tr.vperf,':',tr.epoch,tr.tperf,'-.')
legend('Training','Validation','Test',-1);
ylabel('Squared Error'); xlabel('Epoch');
```

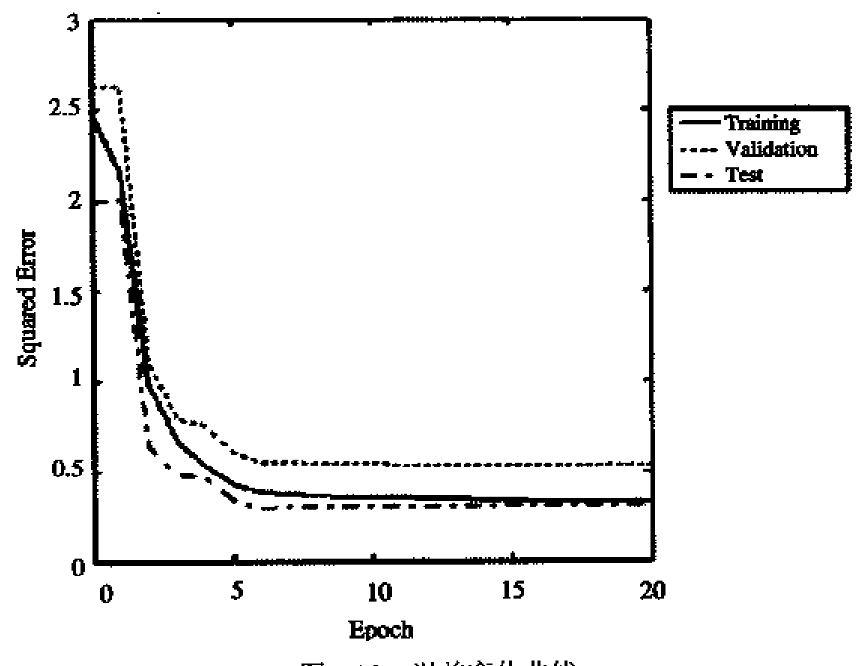


图 4.50 误差变化曲线

(4) 网络仿真。

为了进一步检验训练后网络的性能，下面对训练结果作进一步仿真分析。利用 postreg 函数可以对网络仿真的输出结果和目标输出作线性回归分析，并得到两者的相关系数，从而可以作为网络训练结果优劣的判别依据。仿真与线性回归分析如下：

```
an = sim(net,ptrans);
a = poststd(an,meant,stdt);
```



```
for i=1:3
    figure(i)
    [m(i),b(i),r(i)] = postreg(a(i,:),t(i,:));
end
```

图 4.51～图 4.53 分别给出了网络三个输出的线性回归分析结果曲线。

由图 4.51 和图 4.52 回归分析结果可见，hdl 和 ldl 指标值的检测结果输出是十分满意的，它们与目标输出的相关系数几乎达到了 0.9。由图 4.53 可见，网络 vldl 指标值的输出结果与期望值偏差较大，需要重新对网络进行设置和训练。有兴趣的读者可以适当增加 BP 网络中间层的神经元个数，或采用贝叶斯正则化方法，重新对网络进行训练，以期得到更好的网络性能。

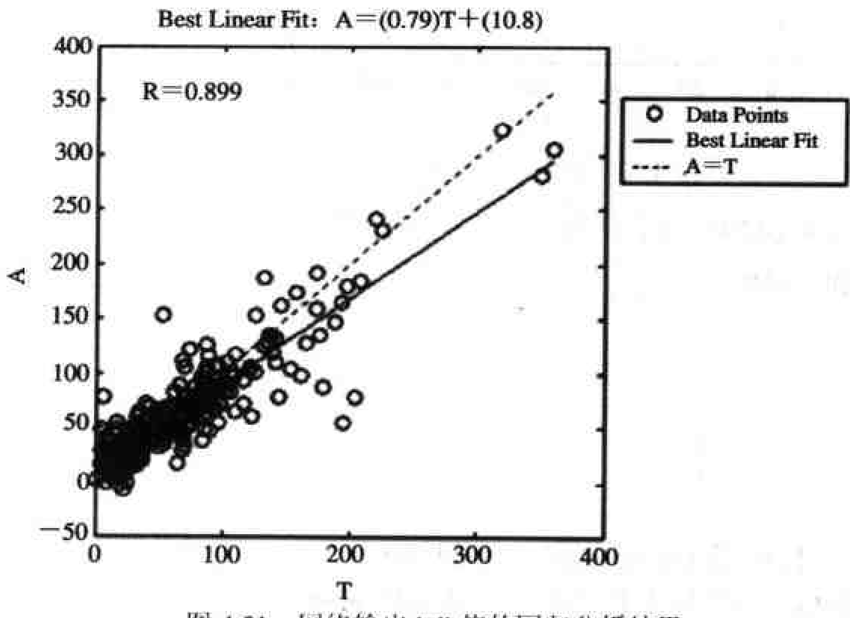


图 4.51　网络输出 hdl 值的回归分析结果

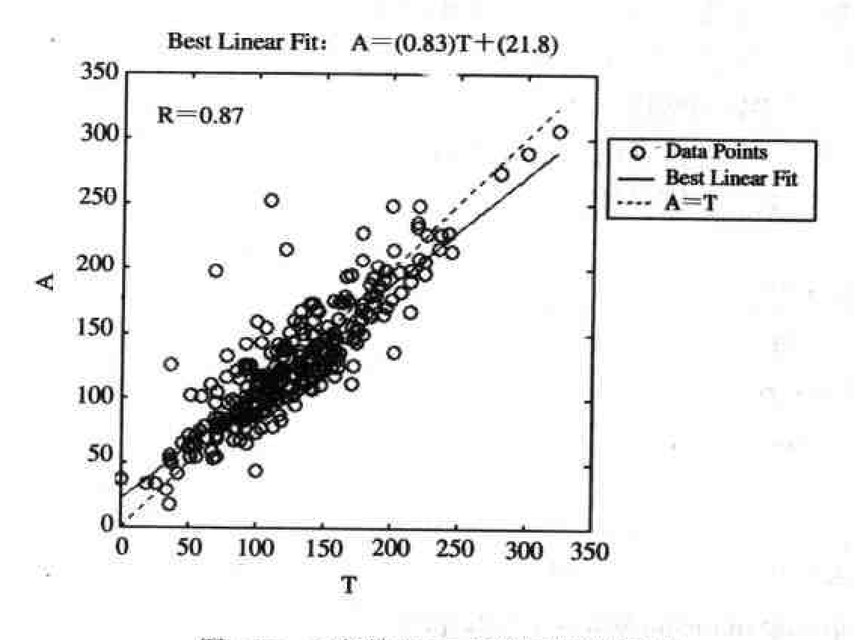
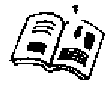


图 4.52　网络输出 ldl 值的回归分析结果



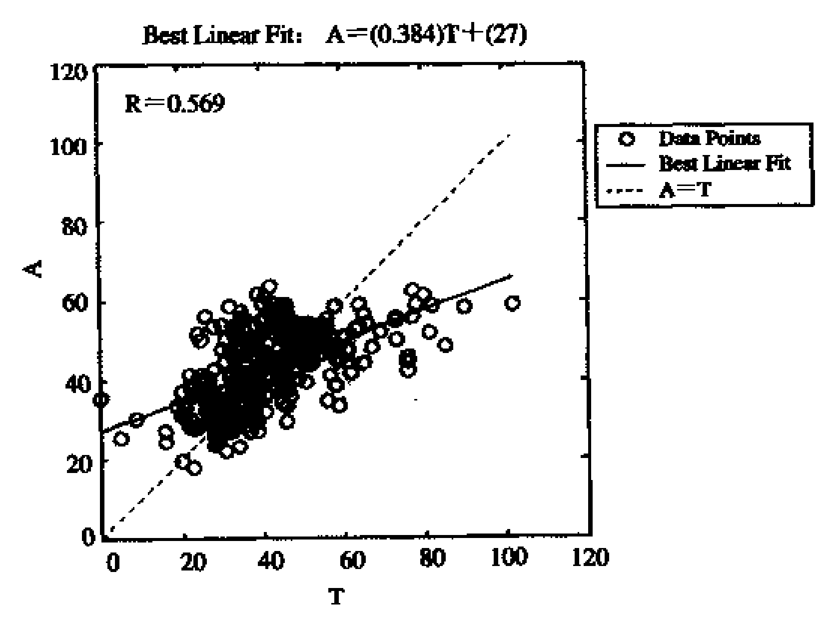


图 4.53 网络输出 vldl 值的回归分析结果

(4) 给出本例的 MATLAB 程序：

```
% Example 4.26
%
close all
clear
echo on
clc
% BP 神经网络用于血清胆固醇含量检测
% PRESTD——对样本数据进行标准化处理
% PREPCA——对样本数据进行主元分析
% NEWFF——生成一个新的前向神经网络
% TRAIN——对 BP 网络进行训练
% SIM——对 BP 网络进行仿真
% POSTREG——对仿真结果进行回归分析
pause    % 敲任意键开始
clc
% 加载样本数据
load choles_all
sizeofp = size(p)
sizeoft = size(t)
pause
clc
% 对样本数据进行标准化处理
[pn,meanp,stdp,tn,meant,stdt] = prestd(p,t);
```



```
% 对样本数据进行主元分析
[ptrans,transMat] = prepca(pn,0.001);
[R,Q] = size(ptrans)
pause
clc
% 将样本数据划分为训练集、验证集和测试集
iitst = 2:4:Q;
iival = 4:4:Q;
iitr = [1:4:Q 3:4:Q];
val.P = ptrans(:,iival); val.T = tn(:,iival);      % 验证样本集
test.P = ptrans(:,iitst); test.T = tn(:,iitst);    % 测试样本集
ptr = ptrans(:,iitr); ttr = tn(:,iitr);            % 训练样本集
pause
clc
% 创建神经网络
net = newff(minmax(ptr),[5 3],{'tansig' 'purelin'},'trainlm');
pause
clc
% 对神经网络进行训练
[net,tr]=train(net,ptr,ttr,[],[],val,test);
% 绘制误差变化曲线
plot(tr.epoch,tr.perf,tr.epoch,tr.vperf,':',tr.epoch,tr.tperf,'-.')
legend('Training','Validation','Test',-1);
ylabel('Squared Error'); xlabel('Epoch');
pause
clc
% 对网络进行仿真分析
an = sim(net,ptrans);
a = poststd(an,meant,stdt);
% 将仿真结果与目标输出作线性回归分析
for i=1:3
    figure(i)
    [m(i),b(i),r(i)] = postreg(a(i,:),t(i,:));
end
pause
echo off
```

例 4.27　利用神经网络进行字符识别。字符识别是模式识别的一个典型应用，教会计算机识别字符，在某些领域，比如银行支票处理方面，可以大大提高支票的处理效率。在本例中，我们将训练 BP 神经网络使其能够对 26 个英文字母进行识别。





(1) 问题描述。

在计算机中，字符或图像均可以用位图形式加以描述。图 4.54 给出了字母 A 的位图形式，如果☐用 1 表示，其余元素点用 0 表示，则字母 A 可以用一个 7×5 的矩阵来表示。

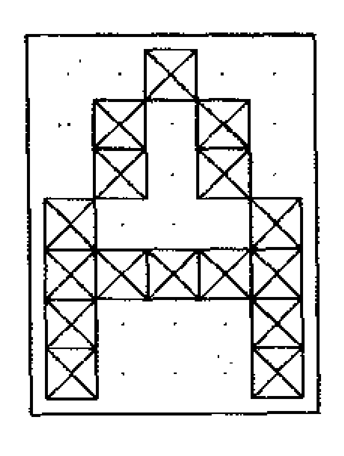


图 4.54 字母 A 的位图

$$A=\begin{bmatrix}0&0&1&0&0\\0&1&0&1&0\\0&1&0&1&0\\1&0&0&0&1\\1&1&1&1&1\\1&0&0&0&1\\1&0&0&0&1\end{bmatrix}$$

如果将上述矩阵按行拉直，则可以得到以列矢量形式表示的字母 A，即

$$A=[00100010100101010001111111000110001]^{T}$$

该列矢量可以作为识别字母 A 的特征矢量。以同样的方式，可以定义其他所有英文字母的特征矢量，以作为神经网络的识别依据（识别网络输入）。

神经网络的识别结果（目标输出），即 26 个英文字母的模式矢量，可以用一个 26 维的列矢量表示，比如字母 A、B 的模式矢量分别为

$$target_A=[10000000000000000000000000]^{T}$$

$$target_B=[01000000000000000000000000]^{T}$$

其他字母依此类推。

所有字母的理想特征矢量和模式矢量即构成了神经网络的训练样本集。通过调用 prprob 函数可以自动生成上述样本集，即

```
[alphabet,targets] = prprob;
[R,Q] = size(alphabet)
```

其中，alphabet 为所有字母的理想特征矢量集合，targets 为相应的模式矢量集合。

可见，利用神经网络进行字符识别，实际上是通过训练神经网络，使其能够根据字符的特征输入得到期望模式矢量的过程。然而，在实际识别过程中，字符的特征输入矢量中可能会混入噪声，例如，图 4.55 给出了一个含有噪声的字母 A 的位图，因此，在训练和设计神经网络时，应该能够使网络具有一定的抑制噪声的能力。

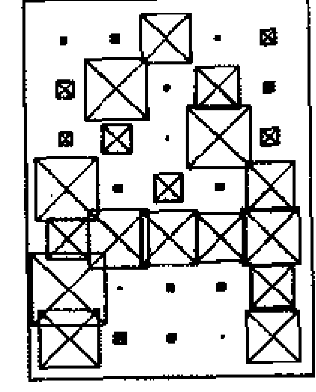


图 4.55 含有噪声的字母 A 位图

(2) 网络的生成与训练。

在神经网络设计过程中，我们采用如图 4.56 所示的前向网络结构。其中，网络输入的个数即为字符特征矢量的维数 35，输出神经元的个数即为字符模式矢量的维数 26，隐层



神经元个数暂时取为 10；隐层和输出层神经元的传递函数均取 logsig 函数，这样可以保证网络输出 0～1 范围内的数；网络训练函数取 traingdx。

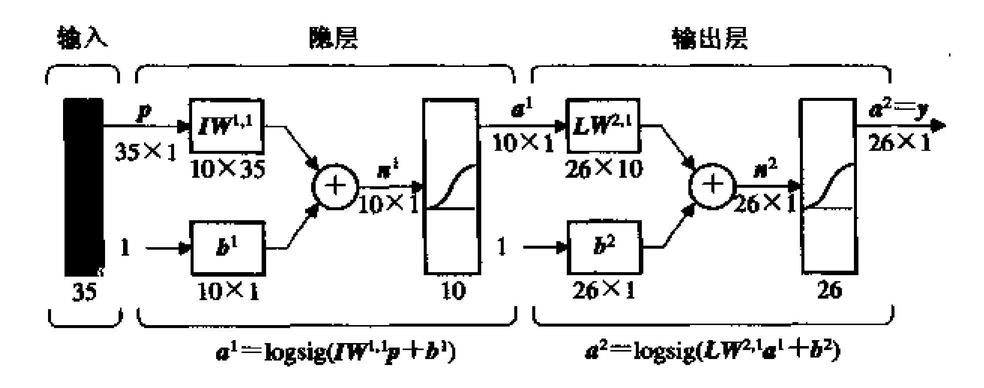


图 4.56 识别网络的结构

上述神经网络可以通过如下语句来生成：

```
S1 = 10;
[R,Q] = size(alphabet);
[S2,Q] = size(targets);
P = alphabet;
net = newff(minmax(P),[S1 S2],{'logsig' 'logsig'},'traingdx');
```

为了提高神经网络抑制噪声的能力，我们采用理想样本数据和含有不同程度噪声的样本数据同时对网络进行训练。训练过程分以下三步进行：

① 用理想样本数据训练神经网络。

```
P = alphabet;
T = targets;
net.performFcn = 'sse';
net.trainParam.goal = 0.1;
net.trainParam.show = 20;
net.trainParam.epochs = 5000;
net.trainParam.mc = 0.95;
[net,tr] = train(net,P,T);
```

② 用含有噪声的样本数据训练神经网络。

```
netn = net;
netn.trainParam.goal = 0.6;
netn.trainParam.epochs = 300;
T = [targets targets targets targets];
for pass = 1:10
P = [alphabet, alphabet, (alphabet + randn(R,Q)*0.1), (alphabet + randn(R,Q)*0.2)];
[netn,tr] = train(netn,P,T);
end
```



③ 再次用理想的样本数据训练神经网络（同步骤①）。

在网络训练完成之后，由于受噪声的影响，网络输出模式矢量中的元素可能不是单纯的取 0 或 1 两个值，因此，我们将网络的输出通过竞争传递函数 compet 进行运算，从而能够得出最接近网络输出的标准模式矢量。

(3) 网络测试。

现采用含有不同均方差白噪声的输入模式对所设计的神经网络模式识别系统进行仿真，根据仿真结果计算误识率，从而检验识别网络系统的性能。

采用噪声取均值为 0，均方差依次为 0:0.05:0.5 的白噪声模型进行系统仿真。对于每一种不同方差的白噪声，在原有理想训练样本的基础上，我们均产生 100 组含有该类型噪声的样本，然后利用这些样本对识别网络进行仿真，根据仿真结果计算出 100 组样本的误识率。依此类推，最终可以得到如图 4.57 所示的网络误识率与噪声方差的变化关系曲线。

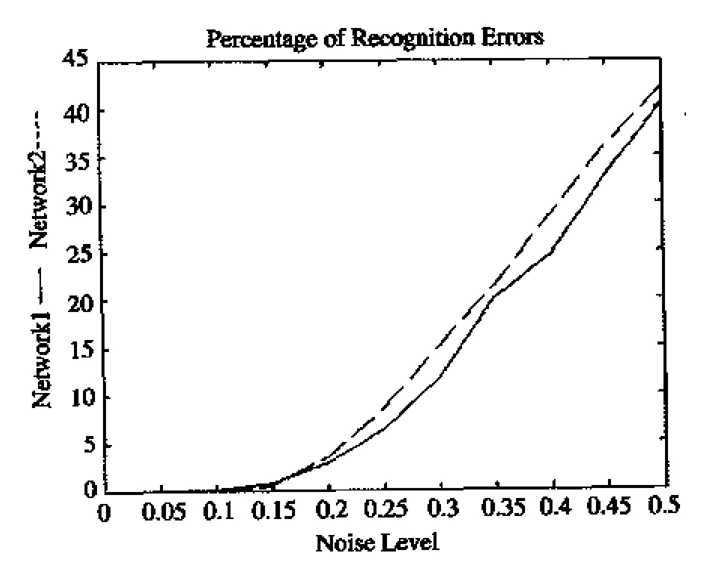


图 4.57 误识率与噪声方差的变化关系曲线

在图 4.57 中，实线和虚线分别代表两种不同识别网络的误识率变化曲线。其中，实线对应的识别网络是经过理想样本和噪声样本集同时训练后得到的结果，而虚线对应的识别网络则只经过了理想样本的训练。由此可见，利用含噪声的样本集对网络进行训练可以大大提高网络的识别能力。同时可以看到，随着噪声均方差的逐步变大，两种网络的误识率均呈上升趋势，而在噪声均方差小于 0.15 之前，网络误识率是很小的。

为了进一步提高网络的识别能力，减小误识率，可以通过进一步增加含噪声的训练样本集的大小或增大识别网络的规模来实现，有兴趣的读者不妨一试。

为了更形象地演示神经网络的识别过程，我们可以用上面设计的神经网络对一个含有噪声的字母 B 进行识别，其 MATLAB 语句如下：

```
noisyB = alphabet(:,2)+randn(35,1) * 0.2;
plotchar(noisyB);
A2 = sim(net,noisyB);
A2 = compet(A2);
```



```
answer = find(compet(A2) == 1);
plotchar(alphabet(:,answer));
```

噪声字母 B 及最终识别模式的位图分别由图 4.58 和图 4.59 给出。由图可见，识别网络正确地识别出了字母 B。

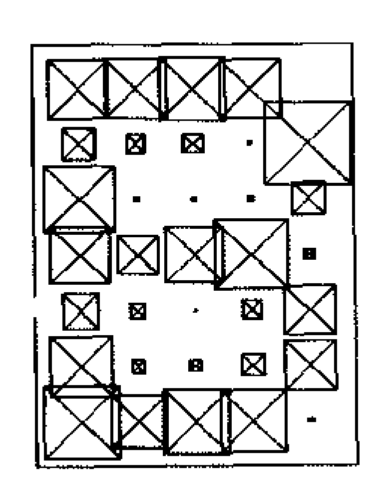


图 4.58 含有噪声的字母 B 的位图

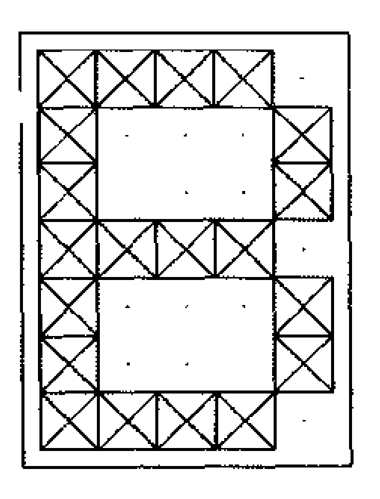


图 4.59 识别结果

(4) 给出本例的 MATLAB 程序：

```
% Example 4.27
%
close all
clear
echo on
clc
% 用 BP 神经网络进行英文字符识别
% NEWFF——生成一个新的前向神经网络
% TRAIN——对神经网络进行训练
% SIM——对神经网络进行仿真
pause      % 敲任意键开始
clc
% 载入训练样本
 [alphabet,targets] = prprob;
[R1,Q1] = size(alphabet)
[R2,Q2] = size(targets)
pause
clc
% 生成神经网络
S1 = 10;
S2 = R2;
net = newff(minmax(alphabet),[S1 S2],{'logsig' 'logsig'},'traingdx');
```



```
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;
pause
clc
% 训练神经网络
% 1.用理想样本训练神经网络
P = alphabet;
T = targets;
net.performFcn = 'sse';
net.trainParam.goal = 0.1;
net.trainParam.show = 20;
net.trainParam.epochs = 5000;
net.trainParam.mc = 0.95;
[net,tr] = train(net,P,T);
pause     % 敲任意键继续
clc
% 2.用含有不同程度噪声的样本训练神经网络
netn = net;
netn.trainParam.goal = 0.6;
netn.trainParam.epochs = 300;
T = [targets targets targets targets];
for pass = 1:10
      fprintf('Pass = %.0f\n',pass);
      P = [alphabet, alphabet, ...
         (alphabet + randn(R,Q)*0.1), ...
         (alphabet + randn(R,Q)*0.2)];
      [netn,tr] = train(netn,P,T);
      echo off
end
echo on
pause     % 敲任意键继续
clc
% 3.再次用理想样本训练神经网络
netn.trainParam.goal = 0.1;
netn.trainParam.epochs = 500;
netn.trainParam.show = 5;
P = alphabet;
T = targets;
[netn,tr] = train(netn,P,T);
% 训练结束
```



```
pause
clc
% 对识别网络进行测试
% 设置测试参数
noise_range = 0:.05:.5;    % 噪声均方差
max_test = 100;
network1 = [];    % 网络 1 的误识率
network2 = [];    % 网络 2 的误识率
% 进行网络仿真和测试
for  noiselevel = noise_range
  fprintf('Testing networks with noise level of %.2f.\n',noiselevel);
  errors1 = 0;
  errors2 = 0;
  for i=1:max_test
      P = alphabet + randn(35,26)*noiselevel;
      % 测试网络 1
      A = sim(net,P);
      AA = compet(A);
      errors1 = errors1 + sum(sum(abs(AA-T)))/2;
      % 测试网络 2
      An = sim(netn,P);
      AAn = compet(An);
      errors2 = errors2 + sum(sum(abs(AAn-T)))/2;
      echo off
   end
   network1 = [network1 errors1/26/100];
   network2 = [network2 errors2/26/100];
end
echo on
pause    % 敲任意键绘制测试结果曲线——误识率曲线
clc
% 绘制网络误识率曲线
plot(noise_range,network1*100,'--',noise_range,network2*100);
title('Percentage of Recognition Errors(%)');
xlabel('Noise Level');
ylabel('Network 1 - -    Network 2 ---');
pause
clc
% 一个测试实例
% 生成一个含噪声的字母 B 作为网络输入
```



```
noisyB = alphabet(:,2)+randn(35,1) * 0.2;
% 绘制含噪声的字母 B 的位图
plotchar(noisyB);
pause
clc
% 对输入模式进行识别
A2 = sim(net,noisyB);
A2 = compet(A2);
answer = find(compet(A2) == 1);
% 绘制识别结果对应的模式位图
plotchar(alphabet(:,answer));
pause
echo off
```

4.7.4 神经网络在预测控制中的应用

预测控制是 20 世纪 70 年代后期发展起来的一类新型计算机控制算法，这种算法的本质特征主要包含三个要素：预测模型、滚动优化和反馈校正。目前，线性系统的预测控制问题已经得到了解决，但对于非线性系统的预测控制问题，由于要面临建立非线性系统的预测模型的问题，所以解决起来就相对困难得多。由于神经网络可以对非线性动态过程进行精确的描述，因此，利用神经网络进行预测控制可以较好地解决非线性系统的预测控制问题。

MATLAB 6.x 的动态仿真工具 Simulink 中提供了专门用于设计神经网络预测控制器的工具模块。在建立了被控对象的系统模型后，利用该模块，用户可以方便地建立基于神经网络的预测控制系统，还可以在 Simulink 环境中对所设计的系统进行动态可视化仿真和分析。本节我们将参照 MATLAB 自身所提供的一个应用实例，介绍 Simulink 环境下神经网络预测控制器的设计和仿真过程。

例 4.28 设计神经网络预测控制器对非线性系统进行控制。如图 4.60 所示为某一溶液搅拌系统，其中，w_1、w_2 分别为两个入口溶液的流速，C_{b1}、C_{b2} 为相应入口溶液的浓度，w_0 为出口溶液的流速，C_b 为出口溶液的浓度，h 表示搅拌容器中溶液的高度。系统模型可以用如下方程式来描述：

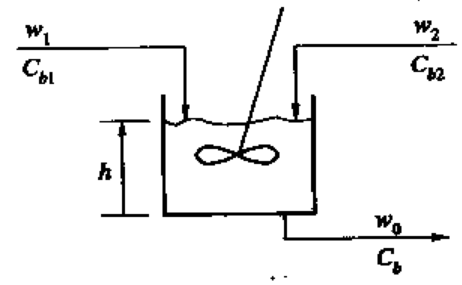


图 4.60 溶液搅拌系统

$$\frac{\mathrm{d}h(t)}{\mathrm{d}t}=w_1(t)+w_2(t)-0.2\sqrt{h(t)}$$

$$\frac{\mathrm{d}C_b(t)}{\mathrm{d}t}=(C_{b1}-C_b(t))\frac{w_1(t)}{h(t)}+(C_{b2}-C_b(t))\frac{w_2(t)}{h(t)}-\frac{k_1C_b(t)}{(1+k_2C_b(t))^2}$$



在上述公式中，设 $w_1=0.1$，$C_{b1}=24.9$，$C_{b2}=0.1$，$k_1=k_2=1$，要解决的问题是通过控制入口处溶液二的流速 w_2，使得出口溶液的浓度 C_b 保持恒定。

(1) 神经网络预测控制的基本原理。

在进行系统设计以前，首先简要介绍一下神经网络预测控制系统的基本结构和原理。神经网络预测控制系统的基本结构如图 4.61 所示。

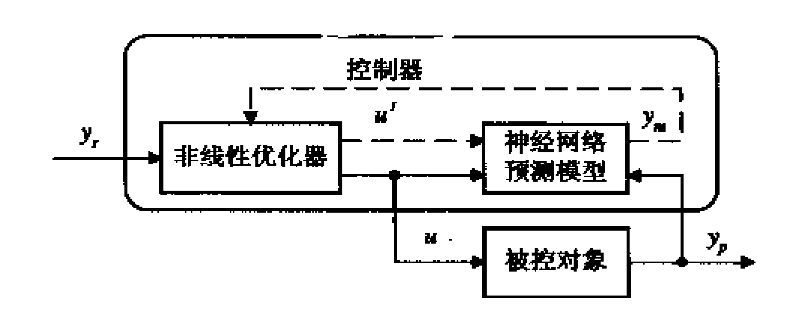


图 4.61　神经网络预测控制系统的结构

首先，神经网络预测模型主要用来描述被控对象的动态行为，它根据系统当前的控制输入和输出信息预测未来的输出值。构建神经网络预测模型是设计神经网络预测控制器的第一步，它通常可以根据从被控对象所获取的输入 / 输出样本数据，对神经网络进行离线训练生成。用作预测模型的神经网络可以采用如图 4.62 所示的带有输入延迟链的网络结构。

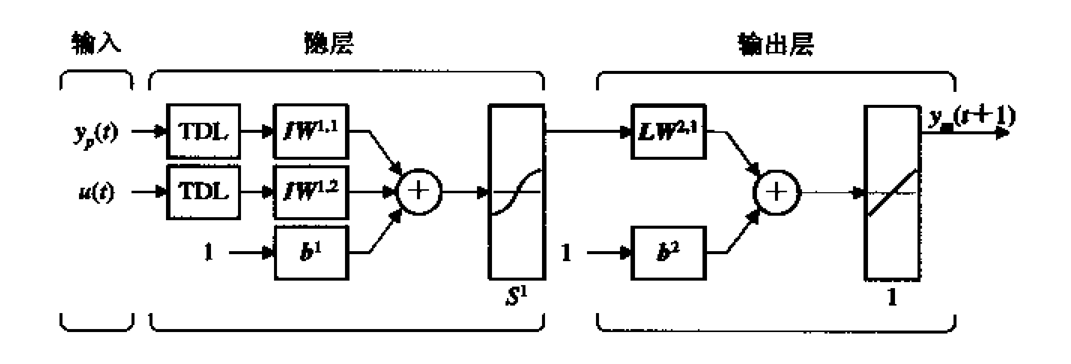


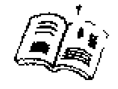
图 4.62　用作预测模型的神经网络结构

其次是非线性优化器，这是预测控制器的核心部分。非线性优化器即通过优化如下性能指标函数来计算系统的控制输入信号：

$$J(N_1,N_2,N_u)=\sum_{j=N_1}^{N_2}e(t+j)^2+p\sum_{j=1}^{N_u}\Delta u'(t+j-1)^2$$

其中

$$e(t+j)=y_r(t+j)-y_m(t+j)$$



为未来时刻系统期望输出与预报输出的误差。

$$\Delta u'(t+j-1)=u'(t+j-1)-u'(t+j-2)$$

为未来时刻的控制增量。优化指标中，N_1 称为最小输出预报区间；N_2 称为最大输出预报区间，表明了待优化的未来输出需要被跟踪的时间范围；N_u 是控制长度，表示未来要纳入考虑的时间控制范围；p 为加权因子，表示控制能量对优化指标的贡献度。所有这些优化参数在设计控制器时均可自行设定。

关于神经网络预测控制的有关知识，读者可参考相关的专业文献，这里不再详述。

(2) 控制系统的设计与仿真。

下面，我们利用 Simulink 动态仿真工具和相关模块设计神经网络预测控制系统，用于对溶液搅拌系统的流出溶液浓度进行控制。控制系统的设计过程主要分以下几步进行：

① 构建被控对象——溶液搅拌系统模型；

② 构建神经网络预测模型；

③ 设置非线性优化器的优化参数；

④ 建立反馈控制系统模型，对系统进行仿真。

启动 MATLAB，在命令窗口中键入命令 predcstr 即可进入如图 4.63 所示的已经构建好的控制系统模型窗口。其中，NN Predictive Controller 模块为神经网络预测控制器模块，Plant 模块即为溶液搅拌系统模型。

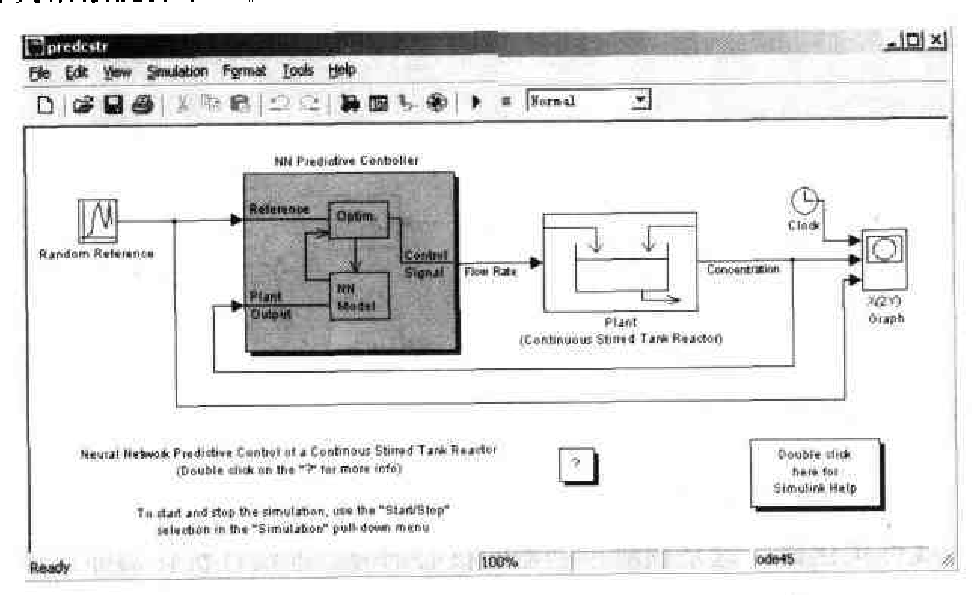


图 4.63　溶液搅拌器神经网络预测控制系统模型

双击 Plant 模块，即可打开如图 4.64 所示溶液搅拌系统的 Simulink 模型，该系统为一个单输入、单输出系统，输入为控制量 w_1，输出为流出溶液的浓度 C_b。溶液搅拌系统的 Simulink 模型文件名为 cstr.mdl。

双击 NN Predictive Controller 模块，即可显示如图 4.65 所示的神经网络预测控制器设计界面。利用该界面，可以对预测优化算法中所用到的优化参数进行设置。



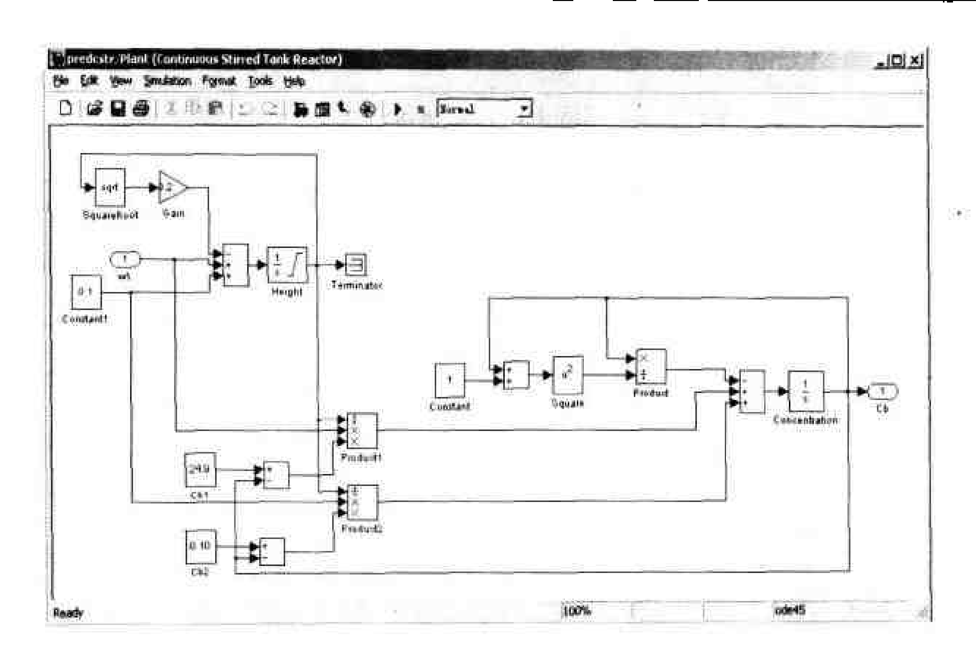


图 4.64 溶液搅拌系统的 Simulink 模型

在图 4.65 中，点击 Plant Identification 按钮，即可进入神经网络预测模型设计界面，如图 4.66 所示。在编写预测优化算法之前，首先要对神经网络预测模型进行设计。神经网络预测模型的设计过程即为神经网络的训练过程。利用如图 4.66 所示界面，可以方便地对用作预测模型的神经网络的结构进行设置，利用已构建好的被控对象模型生成神经网络训练样本数据，设置训练参数，最后完成对神经网络的训练。

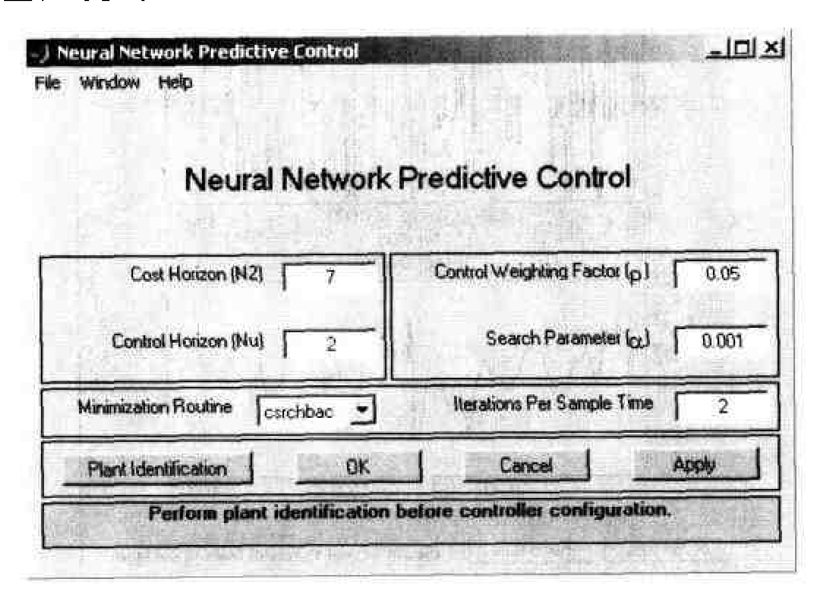


图 4.65 神经网络预测控制器的参数设置界面

在训练神经网络之前，首先须生成训练样本数据。点击 Generate Training Data 按钮，系统将根据设定的训练数据的要求产生一系列白噪声作为激励，通过已定义好的被控对象产生相应的输出数据。图 4.67 给出了某次生成的训练样本数据。若生成的数据满意，则点击 Accept Data 按钮并返回如图 4.66 所示的界面。





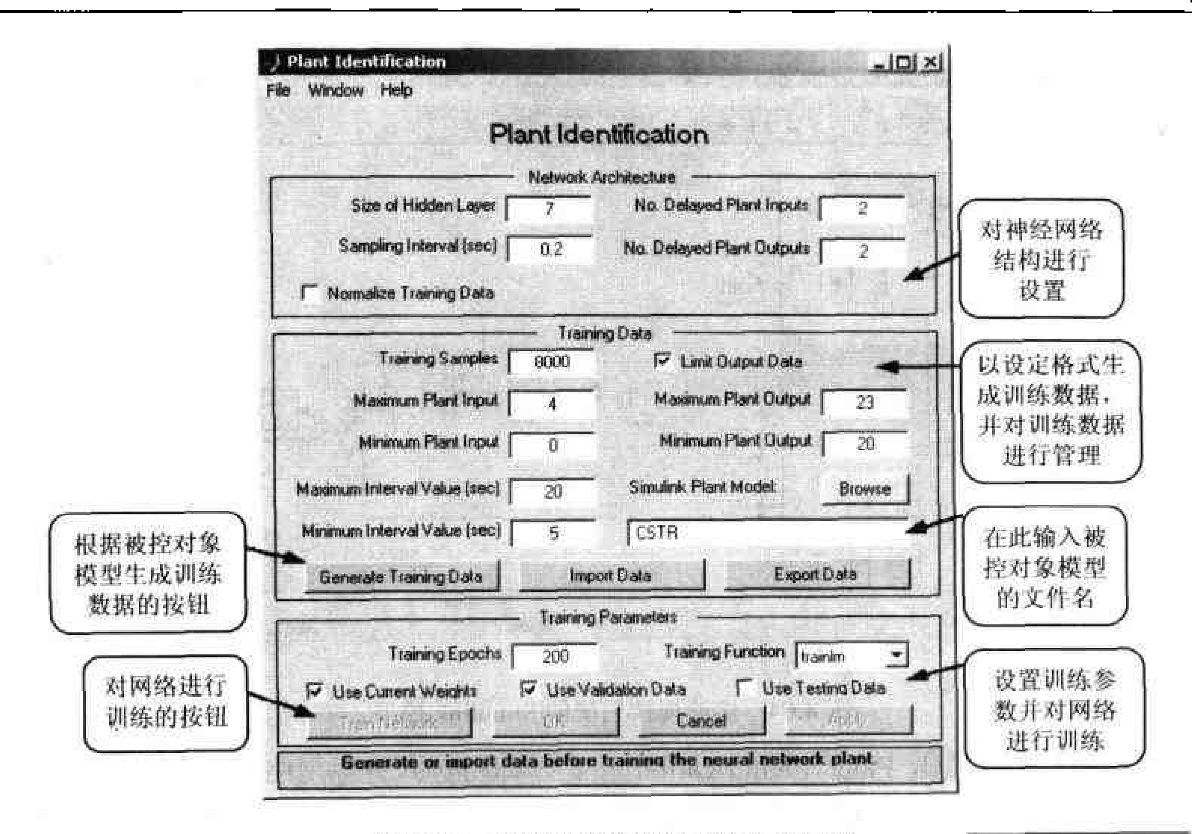


图 4.66 神经网络预测模型的设计界面

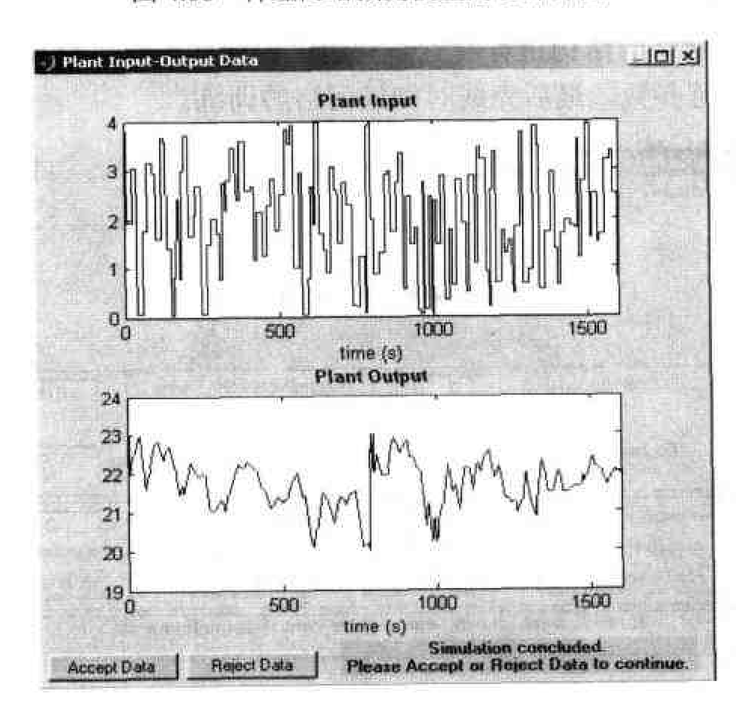


图 4.67 生成的训练样本数据

设置好训练参数后，点击 Train Network 按钮即可对配置好的神经网络进行训练。训练完成后，显示如图 4.68 所示的神经网络预测模型的输出曲线（同时显示的曲线还有训练误差变化曲线，若训练前选择了验证样本或测试样本，还会给出神经网络预测模型对这两个样本输入集的响应曲线）。训练完成后，可以在此基础上重新生成一组训练数据，再次对网络进行训练。点击 OK 按钮，则完成神经网络预测模型的设计，同时返回主界面。



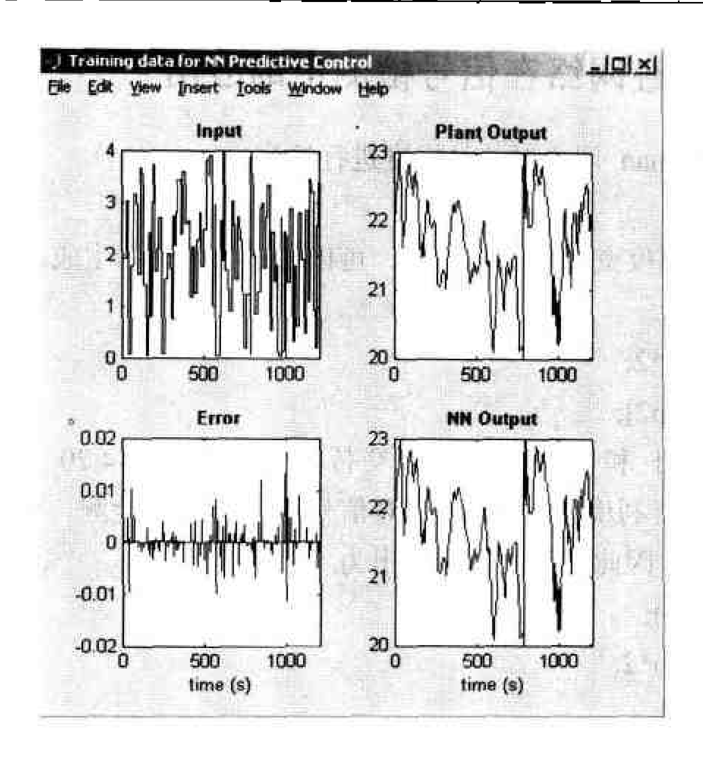


图 4.68 训练结果曲线

在返回如图 4.65 所示主界面后，即可对非线性优化器的优化参数进行设置。设置的优化参数如图 4.65 所示。点击 OK 按钮，则整个神经网络预测控制器的设计完成。

利用设计好的神经网络预测控制器和被控对象构建出的溶液搅拌器神经网络预测控制系统模型如图 4.63 所示。这时，我们可以对设计好的系统进行动态可视化仿真，仿真结果曲线如图 4.69 所示。

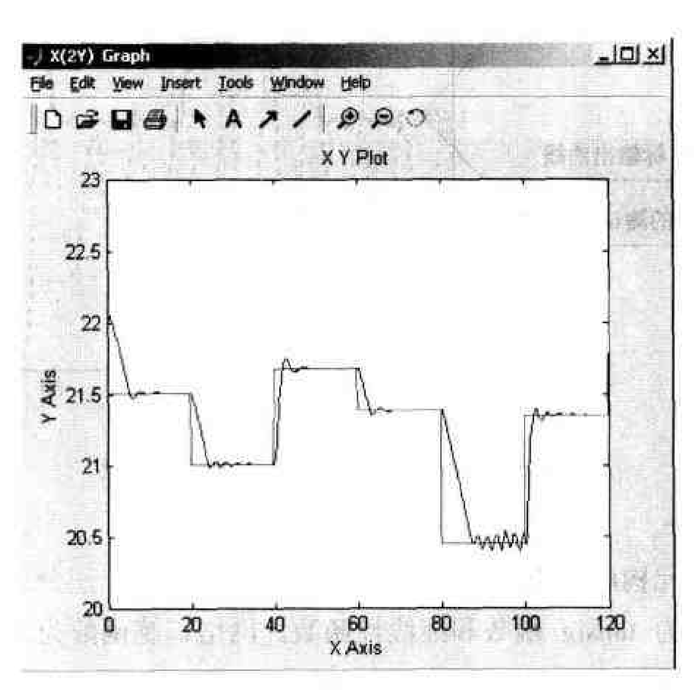
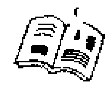


图 4.69 仿真结果曲线



4.7.5 Elman 神经网络在信号检测中的应用

例 4.29 利用 Elman 网络对信号幅度进行检测。

(1) 问题描述。

待检测的信号是幅度变化的正弦信号，可以通过如下语句生成：

```
p1 = sin(1:20);
p2 = sin(1:20)*2;
p = [p1 p2 p1 p2];
```

该信号由幅度分别为 1 和 2 的正弦信号交替变化构成，图 4.70 中所绘的虚线即是待测的正弦信号。本例中要利用 Elman 网络对信号的幅度进行检测，希望网络能够正确地输出时变信号的幅度值，因此网络的目标输出为

```
t1 = ones(1,20);
t2 = ones(1,20)*2;
t = [t1 t2 t1 t2];
```

目标输出曲线如图 4.70 中所绘的短划线。网络的输入和目标输出应采用串行序列格式：

```
Pseq = con2seq (p);
Tseq = con2seq (t);
```

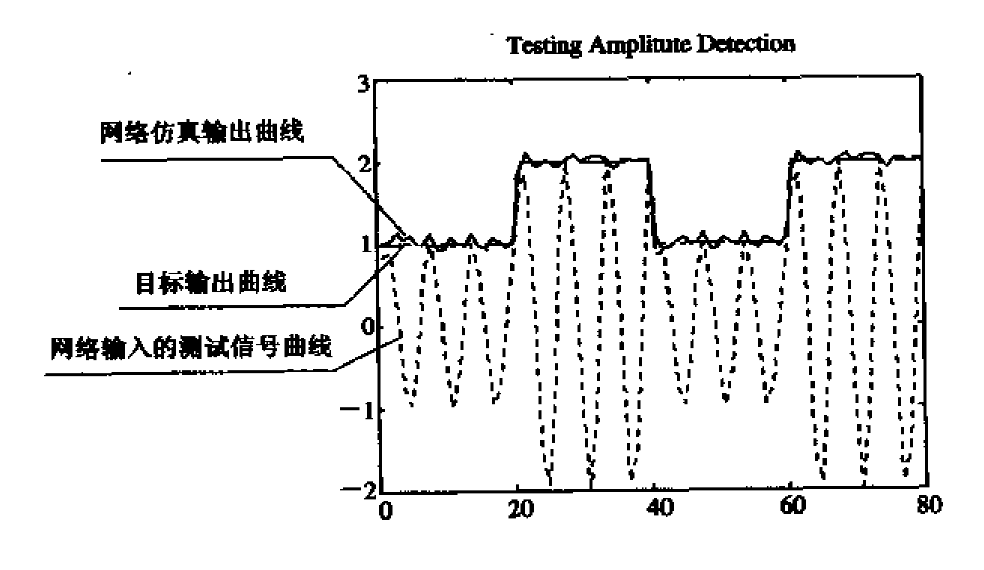


图 4.70　Elman 网络的信号幅度检测性能

(2) 进行网络设计。

建立由两层神经元构成的 Elman 网络，网络的隐层和输出层分别含有 10 个和 1 个神经元，传递函数分别为 tansig 函数和纯线性函数，网络训练函数为 traingdx 函数。

```
net = newelm ([−2 2], [10 1], {'tansig','purelin'}, 'traingdx');
```

(3) 进行网络训练。

设置好网络的训练参数后，就可以对网络进行训练了。






```
net.trainParam.epochs = 1000;          % 训练次数
net.trainParam.show = 20;              % 显示频率
net.trainParam.goal = 0.01;            % 训练目标
net.performFcn = 'sse';                % 性能函数
[net,tr] = train (net, Pseq, Tseq);
```

(4) 对网络进行测试。

利用样本输入数据对训练好的网络进行测试：

```
a = sim (net, Pseq);
```

并绘出样本输入、目标输出和网络仿真输出曲线：

```
time = 1:length(p);
plot (time, p, ':', time, t, '--', time, cat(2, a{:}));
title ('Testing Amplitute Detection');
```

测试结果如图 4.70 所示，图中的虚线、短划线和实线分别表示网络输入的测试信号曲线、目标输出曲线和网络仿真输出曲线。从图 4.70 中可以看出网络对样本信号有较好的检测性能。

(5) 对网络推广性的测试。

利用一组新的输入数据对训练好的网络进行测试。首先产生输入信号和目标输出：

```
p3 = sin(1:20)*2.6;      % 如果输入信号的幅度为 2.6
t3 = ones(1,20)*2.6;     % 那么检测输出应为 2.6
p4 = sin(1:20)*1.2;      % 如果输入信号的幅度为 1.2
t4 = ones(1,20)*1.2;     % 那么检测输出应为 1.2
pg = [p3 , p4 , p3 , p4];
tg = [t3 , t4 , t3 , t4];
pgseq = con2seq(pg);
```

然后利用这一输入信号对网络进行仿真：

```
a = sim (net, pgseq);
```

并绘出测试信号、目标输出和网络仿真输出曲线：

```
figure;
time = 1:length(pg);
plot (time, pg, ':', time,tg,'--',time,cat(2,a{:}));
title ('Testing Generalization');
```

网络推广性能测试结果如图 4.71 所示，图中的虚线、短划线和实线分别表示网络输入的测试信号曲线、目标输出曲线和网络仿真输出曲线。从图 4.71 中可以看出网络对这一测试信号的检测性能虽然不及样本信号，但依然可以较成功地实现幅度检测。



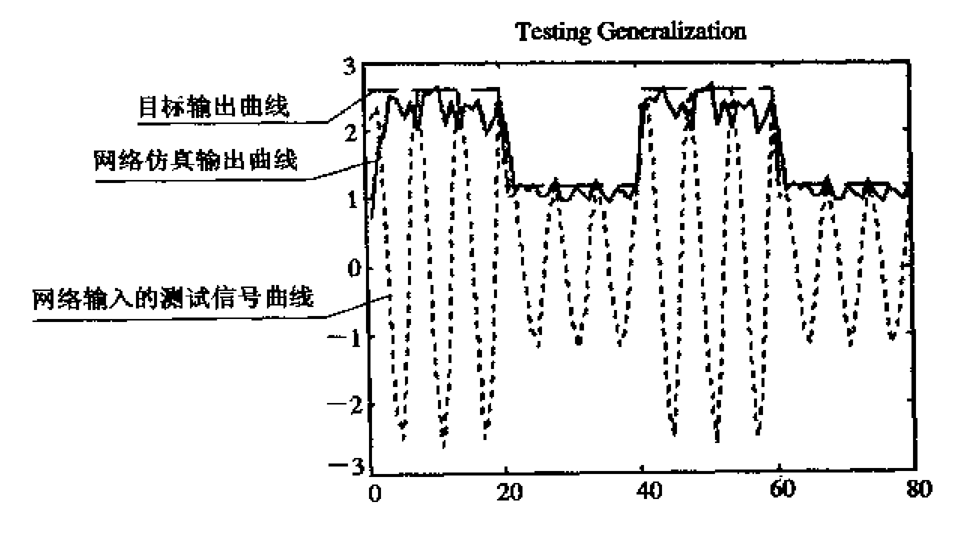


图 4.71 Elman 网络的推广性能

(6) 给出本例的 MATLAB 程序：

```
% Example 4.29
% 产生样本数据 P 和目标输出 T
p1 = sin (1: 20);            % 如果输入信号的幅度为 1
t1 = ones (1, 20);           % 那么检测输出应为 1
p2 = sin(1:20)*2;            % 如果输入信号的幅度为 2
t2 = ones(1,20)*2;           % 那么检测输出应为 2
% 将上述两组数据合并得到网络的输入和目标输出
p = [p1 p2 p1 p2];
t = [t1 t2 t1 t2];
Pseq = con2seq (p);             % 将输入变为串行形式
Tseq = con2seq (t);             % 将目标输出变为串行形式
% 建立 Elman 网络，网络由两层神经元构成
% 两层分别有 10 个和 1 个神经元，传递函数分别为 tansig 函数和纯线性函数
% 训练函数为 traingdx 函数
net = newelm ([-2 2], [10 1], {'tansig','purelin'}, 'traingdx');
% 对网络进行训练
net.trainParam.epochs = 1000;          % 训练次数
net.trainParam.show = 20;              % 显示频率
net.trainParam.goal = 0.01;            % 训练目标
net.performFcn = 'sse';                % 性能函数
[net,tr] = train (net, Pseq, Tseq);
% 利用样本数据对网络进行仿真
a = sim (net, Pseq);
time = 1:length(p);
% 第一幅图：画出样本数据的网络仿真输出图形
plot (time, p, ':', time, t, '--', time, cat(2, a{:}));
```




```
title ('Testing Amplitute Detection');
% 利用一组新数据对网络进行检测
p3 = sin(1:20)*2.6;       % 如果输入信号的幅度为 2.6
t3 = ones(1,20)*2.6;      % 那么检测输出应为 2.6
p4 = sin(1:20)*1.2;       % 如果输入信号的幅度为 1.2
t4 = ones(1,20)*1.2;      % 那么检测输出应为 1.2
% 产生测试数据
pg = [p3 p4 p3 p4];
tg = [t3 t4 t3 t4];
pgseq = con2seq(pg);
% 利用测试数据对网络进行仿真
a = sim (net, pgseq);
% 第二幅图：  画出测试数据的网络仿真输出图形
figure;
time = 1:length(pg);
plot (time, pg, ':', time,tg,'--',time,cat(2,a{:}));
title ('Testing Generalization');
```



附录 常用神经网络工具箱函数索引

参考文献

[1] 楼顺天，施阳编著. 基于 MATLAB 的系统分析与设计——神经网络. 西安: 西安电子科技大学出版社，1998

[2] 张立明编著. 人工神经网络的模型及其应用. 上海: 复旦大学出版社，1993

[3] 王永骥，涂健编著. 神经元网络控制. 北京: 机械工业出版社，1998

[4] The Mathworks Inc. Neural Network Toolbox User's Guide (Version 4). 2001

